NECパーソナルコンピュータ
PC-9800シリーズ

Software Library

MS-DOS® 3.3D

プログラマーズリファレンスマニュアル Vol.2

Software
Library
MS-DOS® 3.3D
プログラマーズリファレンス
マニュアル Vol.2

ご注意

(1) 本書の内容の一部または全部を、無断で他に転載することは禁止されています。
(2) 本書の内容は、将来予告なしに変更することがあります。
(3) 本書の内容は、万全を期して作成しております。万一、ご不審な点や誤り、記載もれなどお気づきの点がありましたら、ご連絡ください。
(4) 運用した結果の影響については、(3)項にかかわらず責任を負いかねますのでご了承ください。

Microsoft（マイクロソフト）とそのロゴは米国マイクロソフト社の登録商標です。
MS-DOSは米国マイクロソフト社の登録商標です。
80286、386、386SX、486、486SXは米国インテル社の商標です。

輸出する際の注意事項

本製品（ソフトウェア）は日本国内仕様であり、外国の規格等には準拠しておりません。
本製品を日本国外で使用された場合、当社は一切責任を負いかねます。また、当社は本製品に関して、海外での保守サービスおよび技術サポート等は行っておりません。

AS1A1A

本書は、「MS–DOSプログラマーズリファレンスマニュアル Vol.1」の続編として、周辺装置を制御するデバイスドライバの技術情報について解説したものです。

本書の目的と構成

本書の目的は、プログラム開発の際に必要なデバイスドライバに関する技術情報について説明しています。したがって、本書をお読みになる場合、プログラム開発に関する基礎的な知識を習得されていることが前提となります。

なお、各デバイスドライバのファンクション部分の見方については、「MS–DOSプログラマーズリファレンスマニュアル Vol.1」を参照してください。

■ 第1章「MS–DOSデバイスドライバ」

この章はオペレーティングシステムを構成するデバイスドライバの種類、構成、作成方法、デバイスファンクションについて説明しています。

また、章の最後にはブロックデハイスドライバ、キャラクタデバイスドライバのプログラム例を掲載しています。

■ 第2章「日本語処理」

日本語処理の機能を制御する、AIかな漢字変換用の2つのデバイスドライバ（NECAIK1.DRV、NECAIK2.DRV）と、単文節変換用のデバイスドライバ（NECDIC.DRV）に関する技術情報について説明しています。

■ 第3章「マウスインターフェイス」

画面上のカーソルの動きを容易にコントロールできるマウス機能を制御するマウス用デバイスドライバ（MOUSE.SYS）に関する技術情報について説明しています。

■ 第4章「グラフィックスドライバ」

グラフィック機能を制御するグラフィックスドライバ（GRAPH.SYS）に関する技術情報について説明しています。

■ 第５章「EMS インターフェイス」

拡張メモリ（1M バイト以上のアドレス空間のメモリ）を制御するデバイスドライバ（EMM.SYS または EMM386.SYS）に関する技術情報について説明しています。

■ 第６章「フォントドライバ」

マルチフォント ROM ボードや本体 ROM などを利用して、2 バイト JIS コードの文字フォントの編集などを制御するフォントドライバ（FONT.SYS）に関する技術情報について説明しています。

その他のマニュアル

「MS–DOS 拡張機能セット」には、本書の他に次のようなマニュアルが添付されています。

■ 『MS–DOS ユーザーズリファレンスマニュアル』

システムディスクに収められている MS–DOS のすべてのコマンドについて、詳しく説明しています。また、「MS–DOS 基本機能セット」では扱われていない、MS–DOS の高度な機能についても解説しています。MS–DOS の手引きとして、ご利用ください。

■ 『日本語入力ガイド』

MS–DOS 上で利用可能な日本語入力機能について解説しています。日本語の入力を行う方法と、その他の有用な機能について詳しく説明し、また、辞書ファイルを保守管理するユーティリティ（DICM）や、ユーザーが独自の記号や漢字を作成して利用するためのユーティリティ（USKCGM）についても説明しています。

■ 『プログラム開発ツールマニュアル』

「MS–DOS プログラム開発ツールディスク」に収められているユーティリティプログラムの、詳細な使用法について解説しています。アセンブリ言語などでプログラムを開発される際に、ご利用ください。

■ 『プログラマーズリファレンスマニュアル Vol.1』

MS–DOS の内部的な技術情報を、詳細に説明しています。MS–DOS の提供する各種機能（システムコール、ファンクションコール）や、プログラムおよびメモリ管理に関する技術情報を扱っています。

目　次

第1章

MS–DOSデバイスドライバ

1.1 デバイスドライバとは

デバイスドライバとは、オペレーティングシステムの構成要素であり、周辺装置(コンピュータが記憶や外部との通信などのために用いるハードウェア)用のコントローラやアダプタを管理するものです。

MS–DOSには、IO.SYSファイル内のドライバ(標準で組み込まれているデバイスドライバ、以後標準デバイスドライバと呼ぶ)に加え、新しいデバイスドライバ（プリンタ、プロッタ、マウスなど）を追加することができます。これはシステムの起動（ブート）時に、参照されるCONFIG.SYSファイル内の〝DEVICE =〞コマンドを使って、デバイスドライバを登録することによって、新規のデバイスドライバを容易に追加することができます。これらのドライバのことをインストール可能なデバイスドライバといいます。

次にそれぞれの、デバイスドライバの構造について説明します。

■ 標準デバイスドライバ

すべてのMS–DOSシステムには、少なくとも5個(コンソール用、シリアルポート用、プリンタポート用、クロック用、記憶デバイス用)の標準のデバイスドライバが存在しています。これらのドライバは、リンクされたリストで示されます。各ドライバの〝ヘッダ〞には、次のドライバに対するDWORDポインタが含まれます。チェイン内の最後のドライバには、–1、–1（全ビットともオン）というエンドオブリストマーカがあります。

チェイン内の各ドライバには、ストラテジと割り込みという2個のエントリポイントがあります。2個のエントリポイントは、MS–DOSの将来のバージョンでマルチタスクをサポートするときのために設けられています。

マルチタスク環境では、入出力を非同期で行わなければなりません。これを実現するには、次のような手順で処理が行われる必要があります。

1. ストラテジルーチンは呼び出されたらリクエストを内部的に待ち行列（キュー）に登録し、即座にリターンする。

2. 割り込みルーチンは割り込みにより起動され、内部的な待ち行列からリクエストを取り出し、それを処理する。

3. リクエストが完了すると、割り込みルーチンは実行済みフラグをセットする。

MS–DOSは、定期的にこの実行済みフラグをチェックし、実行済みフラグのセットされているリクエストを探し、そのリクエストの終了を待っているプロセスに制御を移す。

このようにリクエストを待ち行列に入れる方法では、一時的に複数のリクエストが待ち状態になる可能性があるため、レジスタによってI/O情報を受け渡すことは不可能になります。このため、MS–DOSのデバイスインターフェイスは〝パケット〟を使ってリクエスト情報を渡します。このリクエストパケットは、サイズとフォーマットが可変で、次の2つの部分から構成されます。

- すべてのリクエストで同じフォーマットを持つ静的リクエストヘッダセクション。

- リクエストのタイプごとに固有の情報を持つセクション。

ドライバが呼び出されるとパケットに対するポインタが渡されます。MS–DOSの将来のマルチタスクバージョンでは、このパケットがMS–DOSによって保護されるすべてのI/O待ち行列のグローバルチェインにリンクされます。

現在のMS–DOSのバージョンでは入出力が非同期で行われないため、グローバルやローカルの待ち行列をサポートしていません。一時的に待ち状態となるのは1個のリクエストだけなので、2個のエントリポイントは本来の目的で使用されず、処理は次のようになります。

1. ストラテジルーチンは特定の場所にパケットのアドレスを格納し、MS–DOSにリターンする。

2. 割り込みルーチンはストラテジルーチンの直後に呼び出され、パケットをもとにリクエストを処理する。割り込みルーチンから戻った時点で、リクエストは完了したものと見なす。

■ インストール可能なデバイスドライバ

インストール可能なデバイスドライバは、ファイルの先頭にデバイスヘッダを持つBIN形式（.BIN：コアイメージ）または、EXE形式（.EXE）フォーマットでファイルを作成します。デバイスヘッダのリンクフィールドは、–1、–1に初期設定されていなければなりません（SYSINITがこれをセットします）。複数のデバイスドライバを1つのファイルに納める場合、各デバイスドライバのデバイスヘッダにはリスト内の次のデバイスを指すポインタをセットします。このとき最後のデバイスヘッダは–1、–1で初期設定されていなければなりません。

MS–DOSバージョン3.0以降でのデバイスドライバは、BIN形式、EXE形式のどちらのフォーマットでも使用することができます。ただし、バージョン3.0以前のMS–DOSでは、BIN形式のデバイスドライバしか使用できないため、どちらのバージョンでも動作可能なデバイスドライバを作成するときはBIN形式にする必要があります。

1.2　デバイスドライバの種類

デバイスドライバは全体的に見て、キャラクタデバイスドライバとブロックデバイスドライバの2つに分類されます。デバイスドライバがこの2つのグループのどちらに属するかは、そのデバイスがMS-DOSからどのように見えるかということと、ドライバ自体が用意しなければならない機能によって決定されます。

次にそれぞれのデバイスドライバについて説明します。

■ キャラクタデバイスドライバ

キャラクタデバイスドライバは、一度に1バイトずつ読み書きするデバイスのことをいい、通常ランダムアクセスはできません。コンソール、プリンタ、RS-232Cコミュニケーションポートのようなキャラクタの入出力を扱います。これらの装置には特別の名前（ファイル名）が付けられています（CON、PRN、AUXなど）。ユーザーはこれらの装置の名前を指定することによって、入出力を扱うチャネル（ハンドルまたはFCB）をオープンすることができます。なお、キャラクタデバイスの名前は1つなので複数のユニットを定義することはできません。

■ ブロックデバイスドライバ

ブロックデバイスドライバはシステムの〝ディスクドライブ〞のことで、ブロック単位（通常、物理セクタサイズ単位）でデータの読み書きができるランダムアクセスが可能なデバイスです。ブロックデバイスドライバにはファイル名を付けることができないため、直接オープンすることができません。ブロックデバイスはドライブ名（A：、B：、C：）によって識別されます。

ブロックデバイスはユニットで構成されています。1つのドライバは複数のディスクドライブを処理することができます。たとえばALPHAというデバイスドライバは、A：、B：、C：、D：というドライブを処理することができるとします。これは4つのユニット（0〜3）を定義して、4つのドライブ文字を使用するという意味になります。このようにドライバリスト中のドライブの順番によって、どのユニットがどのドライブ文字と対応するかが決定されます。

たとえば、ALPHAというドライバがデバイスリスト中の先頭にあるブロックドライバで、4つのユニット（0〜3）を定義した場合はドライブ名はA：、B：、C：、D：になり、BETAが2番目にあるブロックドライバで、3つのユニット（0〜2）を定義した場合はドライブ名はE：、F：、G：になります。

ブロックデバイスユニットは理論上63まで使用できますが、ドライブを表す文字が〝Z〞（5AH）までなのでこれを超えることはできません。標準のBIOS内に存在するブロックデバイスドライバ（システムのディスクドライブ）は、すべてブロックデバイスドライバより先に置かれます。

注意　デバイスドライバにはファイルにORG 100H（COMファイルのような）を使用しないでください。デバイスドライバはPSPを使用せずに単にロードされるだけです。したがって、このファイルの起点は0でなければ

ばなりません（ORG 0またはORGステートメントなし）。

1.3　デバイスドライバの作成方法

MS–DOSでデバイスドライバを作成するには、ファイルの開始点にデバイスヘッダが付けられているバイナリファイル（COM形式またはEXE形式のファイル）を作成します。

デバイスドライバの場合、コードの起点は100Hではなく00Hでなければなりません。リンクフィールド（次のデバイスヘッダに対するポインタ）は、このファイル内にデバイスドライバが1つしかない場合は–1、–1でなければなりません。またアトリビュートフィールドおよびエントリポイントは、正しくセットしなければなりません。

キャラクタデバイスの場合、名前フィールドにデバイス名をセットします。この名前にはすべての文字をファイル名として8文字使用することができます。この名前が8文字未満の場合は、スペース（20H）を埋めることによって8文字にします。デバイス名にはコロン（：）は含まれません。〝CON〟は〝CON:〟と同じものですが、これはデフォルトのMS–DOSコマンドインタープリタ（COMMAND.COM）の特性であって、デバイスドライバまたはMS–DOSの特性ではありません。キャラクタデバイス名はすべてこの方法で取り扱われます。

MS–DOSでは標準のデバイスを使用する前に常にデバイスドライバを使用するので、新規のCON（コンソール）デバイスを登録する場合、名前を単に〝CON〟のみに指定してください。このとき新規のCONデバイスのアトリビュート内に、標準入出力デバイスビットをセットすることを忘れないでください。最初に一致するものが現れた時点でデバイスリストの走査が停止するので、このデバイスドライバが優先します。

また、キャラクタデバイスドライバを置き換えるのと同じ方法で、標準のディスクブロックデバイスドライバとデバイスドライバを置き換えることはできません。ブロックデバイスドライバは、IO.SYSのデフォルトのディスクドライバによって直接サポートされないデバイスにのみ使用することができます。

注意　MS–DOSでは、メモリの任意の位置にドライバを登録できるので、アドレスの離れたメモリを参照する場合には注意が必要です。ユーザーのドライバがロードされる場合、常に同じ位置にロードされるとは限りません。

■ デバイスストラテジルーチン

このルーチンはデバイスドライバのサービスリクエストが発生するたびにMS–DOSによって呼び出され、これらのリクエストをデバイス割り込みルーチンによって処理される順序で待ち行列に入れます。

このような待ち行列の機能は、非同期I/Oがサポートされるマルチタスク実行時の環境にあるため非常に重要な機能です。MS–DOSバージョン3.Xでは、この種の機能をサポートしていないので一時にサービスできるのは1つのリクエストだけです。

通常このルーチンは非常に短く、1.12「デバイスドライバ例」では各リクエストを単一のポインタ領域に格納しています。

■ デバイス割り込みルーチン

このルーチンにはリクエストをサービスするためのコードのすべてが含まれます。このルーチンは実質的にハードウェアとインターフェイス（ROM の BIOS コールなど）を取ります。通常このルーチンはサポートされる特定のコマンドコードを処理するための一連のプロシージャ、およびエラーハンドリングルーチンによって構成されています。

詳細は 1.12「デバイスドライバ例」を参照してください。

1.4　デバイスドライバの登録方法

バージョン 2.0 以降の MS–DOS では、新規のデバイスドライバを起動時に自由に登録することが可能です。これは CONFIG.SYS ファイルを読み込むことにより、IO.SYS 内の INIT コードによって実行されます。

MS–DOS は次の方法によってデバイスドライバのコールを行います。

1. ストラテジエントリに対して FAR コールを行う。

2. リクエストヘッダ中のデバイスドライバの情報をストラテジルーチンに渡す。

3. 割り込みエントリに対して FAR コールを行う。

この方法によって将来の MS–DOS のバージョンで、マルチタスク処理をサポートする際に容易に対応できるようになっています。

1.5　デバイスヘッダ

デバイスドライバの開始点にはデバイスヘッダを付ける必要があります。

このデバイスヘッダにはデバイスドライバであることを識別し、エントリポイントを定義してデバイスの各種のアトリビュート（属性）を記述します。OPEN/CLOSE/RM 機能（オープン／クローズ／リムーブ）をサポートしている場合はビット 11 を 1 にします。

デバイスヘッダの内容は次のようになります。

2ワード	次のデバイスへのポインタ（ファイルの中で最後または唯一のドライバの場合、–1をセットする）
1ワード	アトリビュート（属性） ビット15＝0　ブロックデバイス ＝1　キャラクタデバイス ビット15＝1の場合さらに 0＝1　カレント標準入力デバイス 1＝1　カレント標準出力デバイス 2＝1　カレントNULデバイス 3＝1　カレントCLOCKデバイス 4＝1　特殊な装置 5　予約域（0であること） 7〜10　予約域（0であること） ビット14＝IOCTLビット ビット13＝キャラクタデバイス（ビット15＝1）の場合 OUTPUT UNTIL BUSY ＝ブロックデバイス（ビット15＝0）の場合 NON FAT ID ビット12＝予約域　0であること ビット11＝OPEN/CLOSE/RMサポート ビット6＝V3.3ビット
1ワード	デバイスストラテジエントリポイントのポインタ
1ワード	デバイス割り込みエントリポイントのポインタ
8バイト	デバイスファイル名 キャラクタデバイスはデバイス名をセット ブロックデバイスは先頭バイトはユニット数を表す

デバイスエントリポインタはワードを使用します。これらのエントリポインタはこのテーブルをポイントするために、同じセグメント番号からのオフセットでなければなりません。たとえば、XXX：YYYがこのテーブルの開始点をポイントとしている場合、XXX：ストラテジとXXX：割り込みがエントリポイントになります。

■ 次のデバイスへのポインタ

次のデバイスヘッダフィールドに対するポインタは、2ワードのフィールド（オフセット、セグメントの順）でデバイスドライバのロード時に、システムリスト中の次のドライバをポイントするようにMS–DOSによってセットされます。

ファイル内に1つしかデバイスドライバが存在しない場合は、ロードされる前に（ファイルとしてディスク上にあるとき）このフィールドを、–1にセットしなければなりません。

ファイル内に複数のドライバが存在する場合は、2ワードポインタの先頭のワードは次のドライバのデバイスヘッダのオフセットでなければなりません。

注意　COM 形式ファイル内に複数のデバイスドライバが存在する場合、このファイル内の最終ドライバのこのフィールドは–1 にセットされていなければなりません。

■アトリビュート（属性）

アトリビュート（属性）のフィールドは、このドライバが扱うデバイスのタイプを識別します。これらのビットはブロックとキャラクタデバイスを区別するほか、選択されたキャラクタデバイスに対し特殊な取り扱いを行います（アトリビュートワード内のあるビットをあるデバイスのタイプに関して定義する場合は、その他のデバイスのタイプ用のドライバでは、そのビットを 0 にしなければなりません）。

たとえばユーザーが標準入出力用にしたい新規のデバイスドライバを所有しているとします。このドライバを登録するとともに、現在の標準入出力（CON：コンソール）を無効にすることを MS–DOS に通知しなければなりません。この作業はアトリビュート（属性）をセットすることによって行います。そのためには、0 ビット目および 1 ビット目を 1 にセットします（これらのビットは別々の役割を果たします）。同様に新規のクロックデバイスは、この該当するビットをセットすることによって登録することができます。NUL デバイスのアトリビュートは存在しますが、この装置に対して再割り当てを行うことはできません。このアトリビュートで NUL デバイスを使用中かどうかは、MS–DOS が調べることができます。

次に各ビットが示す内容について説明します。

・ブロックデバイス用の NON FAT ID ビットは、BUILD BPB（BIOS パラメータブロック）デバイスコールの動作に影響を与えます。また、この NON FAT ID ビットはキャラクタデバイスでは意味が異なります。このビットはそのデバイスが OUTPUT UNTIL BUSY デバイスコールを実施することを表します。

・IOCTL ビットは、キャラクタデバイスにもブロックデバイスに対しても意味があります。IOCTL ファンクションを使用すれば、デバイスドライバはデバイスドライバ自身の目的のために（たとえば、ボーレート、ストップビットなどをセットするために）、データの転送および取得を行うことができます。渡された情報の処理方法はデバイスによって異なりますが、通常の I/O 要求として処理してはいけません。このビットはファンクションリクエスト 44H の IOCTL システムコール（MS–DOS プログラマーズリファレンス Vol.1 を参照）で、デバイスドライバがコントロール文字列を処理可能かどうかを MS–DOS に通知するものです。
ドライバがコントロール文字列を処理できない場合、最初にこのビットを 0 にセットしなければなりません。これによってデバイスとの間でコントロール文字列の転送、または取得が行われようとした場合（ファンクション 44H によって）、MS–DOS はエラーを返します。コントロール文字列を処理可能なデバイスの場合は IOCTL ビットを 1 にセットします。この種類のデバイスの場合、IOCTL 文字列を転送および取得するために MS–DOS は IOCTL 入出力デバ

イスファンクションコールを行います。

・OPEN/CLOSE/RM ビットはバージョン 3.1 以降の MS–DOS に対し、このドライバがバージョン 3.1 以降の追加機能をサポートするかどうかを通知します。バージョン 3.0 以前で作成したドライバをサポートするためには、そのことを検出する必要があります。このビットはバージョン 3.0 以前では 0 で予約されています。新しいデバイスはすべて、OPEN、CLOSE、および REMOVABLE MEDIA コールをサポートしなければならず、このビットを 1 にセットしなければなりません。バージョン 3.0 以前ではこれらのコールを行わないので、バージョン 3.0 以前のドライバはバージョン 3.1 以降でも互換性が保たれます。

・V3.3 ビットはバージョン 3.3 以降の MS–DOS に対し、このドライバがファンクション 440EH と 440FH による論理ドライブのマップをサポートするかどうかを通知します。このビットはさらにファンクション 440CH とファンクション 440DH のサポートも意味します。

■ ストラテジと割り込みルーチン

これらの 2 つのフィールドは、ストラテジおよび割り込みルーチンのエントリポイントへのポインタです。これらは 1 ワードの値を持っているので、デバイスヘッダの同じセグメント内に存在しなければなりません。

■ デバイスファイル名

これは 8 バイトのフィールドで、キャラクタデバイスの名前またはブロックデバイスのユニット数が入っています。ブロックデバイスの場合はユニット数を先頭のバイトに入れます。

1.6　リクエストヘッダ

MS–DOS はデバイスドライバのコールを行う場合、まず最初にストラテジルーチンを呼び出し、リクエストヘッダと呼ばれるデータ構造のセグメントとオフセットを ES：BX レジスタを介して、ストラテジエントリポイントに渡します。リクエストヘッダは固定長ヘッダで処理に必要なデータが入っています。その後に必要なパラメータをセットしてデバイスドライバを呼び出します。

マシンの状態を保存する（たとえば制御が渡されるときすべてのレジスタを保存し、抜け出すときこれらレジスタを復元する）のは、デバイスドライバの役目です。ストラテジまたは割り込みコールが行われるとき、スタック内には約 20 個のデータを入れるのに十分な領域がありますが、これ以上のスタックを必要とする場合はドライバによって必要なスタックをセットします。

リクエストヘッダの内容は次のようになります。

1バイト	レコードの長さ（このリクエストヘッダとパラメータ領域をバイト単位で表した長さ）
1バイト	ユニットコード（処理に用いるサブユニットただしキャラクタデバイスの場合は意味を持たない）
1バイト	コマンドコード
1ワード	ステータス
8バイト	予約域（2つのDWORDのリンクのための予約域、1つはMS-DOSキューのため、もう1つはデバイスキューのリンク用）

このリクエストヘッダの後に、コマンドコードによって異なるパラメータがセットされます。次にリクエストヘッダの各フィールドについて解説します。

■ レコード長

このフィールドはリクエストヘッダにパラメータ領域を加えたサイズ（バイト単位）です。

■ ユニットコード

ユニットコードフィールドは、ユーザーのデバイスドライバ内のどのユニットに対してリクエストが行われるかを識別するものです。たとえばユーザーのデバイスドライバで3つのユニットが定義されている場合、ユニットコードフィールドの値は、0、1、2になる可能性があります。

■ コマンドコードフィールド

リクエストヘッダ内のコマンドフィールドには、0〜24のコマンドコードを入れることができます。コマンドコードとそのファンクションの対応は次のようになります。

これらのファンクションの詳細な説明は1.7「デバイスドライバファンクション」を参照してください。

コマンドコード	ファンクション
0	INIT
1	MEDIA CHECK（ブロックデバイスドライバのみ）
2	BUILD BPB（ブロックデバイスドライバのみ）
3	IOCTL READ （IOCTL機能を持つデバイスドライバのみ）
4	READ（リード）
5	NON-DESTRUCTIVE READ NO WAIT （キャラクタデバイスのみ）
6	INPUT STATUS（キャラクタデバイスのみ）
7	INPUT FLUSH（キャラクタデバイスのみ）
8	WRITE（ライト）
9	WRITE WITH VERIFY（ライトとベリファイ）

コマンドコード	ファンクション
10	OUTPUT STATUS（キャラクタデバイスのみ）
11	OUTPUT FLUSH（キャラクタデバイスのみ）
12	IOCTL WRITE （IOCTL 機能を持つデバイスドライバのみ）
13	DEVICE OPEN （OPEN/CLOSE/RM 機能を持つデバイスドライバのみ）
14	DEVICE CLOSE （OPEN/CLOSE/RM 機能を持つデバイスドライバのみ）
15	REMOVABLE MEDIA （OPEN/CLOSE/RM 機能を持つブロックデバイスのみ）
16	OUTPUT UNTIL BUSY （ビット 13 をセットしているキャラクタデバイスのみ）
19	Generic IOCTL （V3.3 ビット（ビット 6）が 1 のデバイスのみ）
20	DEINSTALL（キャラクタデバイスのみ）
23	Get Drive Map （V3.3 ビット（ビット 6）が 1 のブロックデバイスのみ）
24	Set Drive Map （V3.3 ビット（ビット 6）が 1 のブロックデバイスのみ）

■ ステータスフィールド

リクエストヘッダのステータスフィールドは、次の図で示すような内容です。

15	14 13 12 11 10	9	8	7 6 5 4 3 2 1 0
ERR	予備領域	BUS	DON	エラーコード（ビット 15 が ON の場合）

ステータスフィールドはドライバ割り込みルーチンに制御が渡されるときは 0 で、ルーチンから戻るときにドライバ側でセットします。

8 ビット目は DONE（処理済）ビットで、セットされた場合は動作が完了したことを意味します。ドライバから抜け出すときに 1 にセットされます。

15 ビット目はエラービットです。セットされた場合は下位 8 ビットがエラーを示します。

エラーの意味は次のとおりです。

エラーコード	意　味
00H	ライトプロテクト（保護）違反
01H	ユニットが無効
02H	ドライブの準備ができていない
03H	コマンドが無効
04H	CRC エラー
05H	ドライブリクエストの長さが不正
06H	シークエラー
07H	メディアが無効
08H	セクタが見つからない
09H	プリンタの用紙切れ
0AH	ライトエラー
0BH	リードエラー
0CH	一般的なエラー
0DH	予備
0EH	予備
0FH	ディスクの交換が不正

9 ビット目は BUSY ビットで、STATUS ファンクションおよび REMOVABLE MEDIA ファンクションによってのみセットされます。

1.7　デバイスドライバファンクション

デバイスドライバは 9 個のファンクション（1 つ以上のファンクションの組み合せ）から構成されます。さらにそのファンクションがそれぞれのコマンドラインで構成される場合があります。

またすべてのストラテジルーチンは、リクエストヘッダをポイントしている ES：BX を使用してコールします。割り込みルーチンはリクエストヘッダのポインタをストラテジルーチンが持つキューから得ます。リクエストヘッダのコマンドコードは、行うべきファンクションとリクエストヘッダに続くデータをドライバに通知します。

ここではコマンドコードの 25 個のファンクションを大きく 9 個に分けて解説します。

注意　すべての 2 ワードポインタは、最初にオフセット、次にセグメントが記憶されています。

■ INIT

コマンドコード＝0
INIT ES：BX→

13バイト	リクエストヘッダ
1バイト	ユニット数
2ワード	エンドアドレス
2ワード	BPB配列に対するポインタ（キャラクタデバイスではセットしない）
1バイト	ブロックデバイス番号

このルーチンはデバイスドライバが登録されるときに1回だけ呼び出されます。INITファンクションは、ドライバが制御するハードウェアデバイスが存在して機能しているかどうかのチェック、必要なハードウェアの初期化(プリンタのリセットなど)、ドライバが必要とする割り込みベクタの取得などを行います。INITルーチンはエンドアドレス（デバイスドライバの常駐の部分の終わりに対するDWORDポインタ）を返さなければなりません。これは1回しか必要のない初期設定コードを削除し、スペースを節約する目的で使用することができます。

ドライバは、ユニット数、エンドアドレス、およびBPB配列に対するポインタをセットします。しかしデバイスドライバに入る時点で（ブロックデバイスにおける）BPB配列に対してドライバによってセットされるDWORDは、このドライバがロードされる原因となったCONFIG.SYSファイル内の行の〝device＝〞の後のキャラクタを指しています。そのためドライバはCONFIG.SYSファイルの起動行を調べて、ドライバに渡すべきパラメータを見つけ出すことができます。この行はリターンまたはラインフィールドで終わります。このデータは読み出し専用であり、これによってドライバはCONFIG.SYSファイルの行を調べて引数を見つけ出すことができます。

```
device＝¥dev¥vt52.sys/1
          ↑
          BPBアドレスは、ここを示します。
```

さらにブロックデバイスに関する場合だけは、このドライバによって定義された最初のユニット（A＝0）に割り当てられるドライブ番号が、ブロックデバイス番号フィールドにセットされます。このデータも読み出し専用になります。

デバイスドライバはエンドアドレスを返さなければなりません。これはドライバより上の最初の使用可能なバイトに対するポインタであり、初期設定のコードを捨てるために使用することができます。

さらに、ブロックデバイスは次の情報を返さなければなりません。

1. ユニット数を返す必要があります。ユニット数はこのデバイスドライバで何

台のドライバをサポートしているかを表した数です。MS–DOS はこれを用いて論理デバイス名を決定します。たとえば登録コールの時点で、現在の最大の論理デバイス文字が F で INIT ファンクションがユニット数として 4 を返す場合、これらのデバイスの論理名は G、H、I、J になります。このマッピングはデバイスリストにおけるドライバの位置と、デバイス上のユニット数（デバイス名フィールドの最初のバイトに格納されている）によって決定されます。

2. BPB（BIOS パラメータブロック）へのワードオフセット（ポインタ）の配列に対する DWORD ポインタを返す必要があります。MS–DOS はデバイスドライバによって渡された BPB を用いて内部構造を作成します。デバイスドライバによって定義される各々のユニットに関して、この配列の中に 1 個ずつのエントリが必要です。この方法を用いることにより、もしすべてのユニットが同じであれば、すべてのポインタが同じ BPB を指すことになりスペースが節約できます。デバイスドライバが 2 個のユニットを定義する場合は、DWORD ポインタは 2 個の 1 ワードオフセットの最初の方を指し、これらのオフセットがさらに BPB を指します。BPB のフォーマットについては 1.7 節中の「BUILD BPB」を参照してください。
   DOS 内部構造はフリーポインタによって指されるバイトを起点として構成されるので、このワードオフセットの配列は（リターンによってセットされるフリーポインタの下側を）保護されなければなりません。定義するセクタサイズは、初期設定中に標準のデバイスドライバ（BIOS）によってセットされる最大セクタサイズ以下でなければなりません。これが異なる場合は初期設定は失敗します。

3. ブロックデバイスの INIT で返すべき最後のデータはメディアディスクリプタバイトです。このバイトは MS–DOS 自身に対しては何の意味も持ちませんが、MS–DOS が特定のドライバユニットに関して現在どのようなパラメータを使用しているかが分かるようにデバイスに渡されます。

ブロックデバイスには、〝ダム〟または〝スマート〟の 2 種類があります。

ダムデバイスは種々の可能なメディアとドライブの組み合わせについて、ユニット（同時に DOS 内部構造）を定義します。たとえばユニット 0＝ドライブ 0 片面、ユニット 1＝ドライブ 0 両面などです。この場合メディアディスクリプタバイトは何の意味も持ちません。

スマートデバイスは 1 つのユニットについて複数のメディアの使用を認めます。この場合 INIT で返される BPB テーブルでは、サポートされる最大のメディアを収容できる十分なスペースが定義されていなければなりません。スマートドライバはメディアディスクリプタバイトを用いて、ユニット内に現在あるメディアに関する情報を渡します。

メディアディスクリプタバイトに関する詳細は、1.8「メディアディスクリプタバイト」を参照してください。

キャラクタ系デバイスドライバを ADDDRV、DELDRV コマンドで動的に登録、削除可能にするにはエンドアドレスに加えて次の情報を返す必要があります。

デバイスドライバは、ドライバに入った時点でセットされている〝BPB 配列に対するポインタ〞のすべてのビットを反転してセットしなおすことにより、ADDDRV コマンドに対して動的な登録、削除機能をサポートしているドライバであることを宣言します。

12345678H（ドライバに入った時点の BPB 配列に対するポインタ）
　　↓　すべてのビットを反転
EDCBA987H（ドライバから返される BPB 配列に対するポインタ）

このことにより DELDRV コマンドにてデバイスドライバを削除する場合に、後述のファンクション 20（DEINSTALL）がコールされるようになります。

キャラクタ系デバイスドライバを動的に登録、削除可能とする場合、次の点に注意してください。

1. そのデバイスドライバがハードウェアのモードなどシステムの情報を変更する可能性がある場合、ファンクション 20（DEINSTALL）で元に戻さなければなりません。そのためこの INIT ファンクションでは、変更する可能性のある情報を自身のメモリ内にセーブしておく必要があります。

2. ADDDRV コマンドは割り込みベクタをすべて専用の領域にセーブしておきます。この内容は DELDRV コマンドによって復旧されるため、デバイスドライバは割り込みベクタの内容をセーブ、リストアする必要はありません。

注意　INIT ファンクション処理中には、MS–DOS のファンクションリクエスト（INT 21H）のファンクション 01H〜0CH、25H、30H、35H のみ使用できます。

■ MEDIA CHECK

コマンドコード＝1
MEDIA CHECK ES：BX→

13 バイト	リクエストヘッダ
1 バイト	BPB からのメディアディスクリプタ
1 バイト	返された値
2 ワード	デバイスアトリビュートフィールドのビット 11 がセットされ、かつ上段の 1 バイトフィールドに〝–1〞が返されたときの、以前のボリューム ID へのポインタ

MEDIA CHECK ファンクションはブロックデバイスドライバでのみ使用され

ます。このファンクションはファイルのリードまたはライト以外（例：オープン、クローズ、削除、およびリネームなど）のドライブアクセスコールが待機しているときに、MS–DOS カーネルによって呼び出されます。そして、その時点でドライブ内のメディアが変更されているかどうかを判定します。ドライバがメディアは変更されていないことを確認できたら（ドアロックまたは他のインターロックメカニズムによる）、MS–DOS は FAT の再読み込みを行わないでディスクアクセスを行います。これによって MS–DOS の処理効率は高められます。

MS–DOS に対するこのようなディスクアクセスコール（ファイルのリードまたはライト以外）が発生すると、次のような事象が発生します。

1. MS–DOS はドライブ文字（ドライブ名のコロンなし）を、特定のブロックデバイスのユニット番号に変換します。

2. デバイスドライバが呼び出され、ディスクが変更されているかどうかを調べるために、そのサブユニットに関するメディアチェックをリクエストします。MS–DOS は以前のメディアディスクリプタバイトを渡します。ドライバによって返される値は次のとおりです。

   メディアが変更されていない場合 …… 1
   変更されたかどうか不明な場合 ……… 0
   メディアが変更された場合 …………… –1
   エラー …………………………………… それ以外の値

   メディアが変更されていない場合、MS–DOS は続いてディスクアクセスを行います。
   変更されたかどうか不明の場合は、ディスクセクタのうち修正済みでこのユニットに関してまだディスクに書き戻されていないものがあれば、MS–DOS はディスクは変更されていないものと見なして処理を続けます。MS–DOS はこのユニットに関する他のバッファを無効にし、BUILD BPB ファンクションを実行します。
   メディアが変更されている場合、MS–DOS は書き込みを待っている修正済みデータのあるバッファも含めて、このユニットに関連するすべてのバッファを無効にして BUILD BPB ファンクションを用いて新しい BIOS パラメータブロックをリクエストします。

3. BPB が返されると、MS–DOS は新しい BPB からドライブに関する内部構造を修正してディレクトリおよび FAT を読み込んだ後、続けてディスクアクセスを行います。

以前のメディアディスクリプタバイトがデバイスドライバに渡される点に注意してください。以前のメディアディスクリプタバイトが新しいものと同じである場合はディスクが変更され、新しいディスクがドライブ内にある可能性がありま

す。したがってそのユニットに関するFAT、ディレクトリおよびメモリ内にバッファされたデータセクタはすべて無効であると考えられます。

ドライバのデバイスアトリビュートワードのビット11が1の場合、ドライバが–1（メディアは変更された）を返すとドライバはDWORDポインタを以前のボリュームIDフィールドにセットします。MS–DOSが〝メディアは変更された〞として、DOSバッファキャッシュの状態に基づくエラーであると判定した場合は、DOSはデバイスのために0FHエラーを発生します。ドライバがボリュームIDサポートを実施していないでビット11をセットしている場合は、文字列〝NO NAME〞に対する静的ポインタを0にセットしなければなりません。

ドアロックがない場合は次のように対処します。

ユーザーがディスクを短い一定の時間（ハードウェアに依存します：たとえば2秒）以内で変更するのは普通は不可能です。そのためディスクアクセスの短い一定の時間以内にMEDIA CHECKが発生した場合は、ドライバは〝1〞すなわち〝メディアは変更されていない〞を報告します。これにより処理効率が著しく改善されます。

注意　返されたBPBの中のメディアディスクリプタバイトが、以前のメディアディスクリプタバイトと同じ場合、MS–DOSはディスクのフォーマットが同じであると見なし（ディスクが変更されているとしても）、ディスクの内部構造を更新するステップを飛ばします。したがってBPBはすべてFAT IDバイトとは無関係に、ユニークなメディアバイトを持っていなければなりません。

■ BUILD BPB

コマンドコード＝2
BUILD BPB ES：BS→

13バイト	リクエストヘッダ
1バイト	BPBからのメディアディスクリプタ
2ワード	転送アドレス（デバイスアトリビュートフィールドのビット13に依存する、1セクタ分のスクラッチスペースまたはFATの最初のセクタへのポインタ）
2ワード	BPBに対するポインタ

BUILD BPBファンクションはブロックデバイスについてのみ使用します。

MEDIA CHECKファンクションの項で述べたように、BUILD BPBファンクションは先行するMEDIA CHECKコールによりディスクが変更されているか、または変更された可能性があることを示された場合に任意の時点で呼び出されます。デバイスドライバはBPBに対するポインタを返さなければなりません。この点はBPBへのワードオフセットの配列に対するポインタが返されるINIT

コールとは異なります。

BUILD BPBコールは1セクタバッファに対するDWORDポインタ（転送アドレス）を入手します。このバッファの内容はアトリビュート（属性）フィールド内のNON FAT IDビット（ビット13）によって決定されます。

このビットが0の場合は、このバッファには最初のFATの最初のセクタが含まれます。FAT IDバイトがこのバッファの最初のバイトです。この場合はドライバはこのバッファを変更してはなりません。この最初のFATセクタは、実際のBPBが返される前に読み取られなければならないので、FATのロケーションはすべての可能なメディアに関して同じでなければなりません。

NON FAT IDビットが1（セット）の場合、ポインタはスクラッチスペース（これは任意の目的で使用可能）の1セクタを指します。BPBの構成方法については1.8「メディアデイスクリプタ」および1.9「メディアディスクリプタテーブル」を参照してください。

MS–DOSバージョン3.1にはドアロック、または他の手段によってディスクの変更された時期を知らせる機能を持つデバイスに対するサポートが含まれます。これはデバイスドライバによって返される新しいエラーコード（MS–DOSバージョン3.1以降でサポートされた：エラー15）です。このエラーは〝ディスクが変更されてはならない時点で変更された〟ことを意味し、ユーザーはボリュームIDを用いて正しいディスクを要求されます。ドライバはリードまたはライト動作でこのエラーを発生させる可能性があります。MS–DOSはドライバがメディアの変更を報告し、MS–DOSバッファキャッシュの中に前のディスクへフラッシュする必要のあるバッファがある場合にMEDIA CHECKでエラーを発生します。

このエラーをサポートするドライバではBUILD BPBファンクションは、ディスクからボリュームIDを読み出すためのトリガとなります。この動作はディスクが正常に変更されたことを表します。ボリュームIDはMS–DOSのFORMATユーティリティによってディスクに入れられ、ボリュームIDアトリビュートを持つディスクのルートディレクトリ内の1つのエントリとなります。ボリュームIDはドライバによってASCIZ文字列として格納されます。

ドライバがボリュームIDを返さなければならないという要件は、他のボリューム管理のスキームがASCIZ文字列を使用している限り、そのスキームを排除するものではありません。NUL（存在しない、またはサポートされていない）ボリュームIDは慣例により次の文字列です。

```
DB        "NO NAME           ", 0
```

■ READ、WRITE、WRITE WITH VERIFY

コマンドコード＝3、4、8、9、12、16

READ or WRITE（IOCTLも含めて）または、OUTPT UNTIL BUSY

ES：BX→

13バイト	リクエストヘッダ
1バイト	BPBからのメディアディスクリプタ
2ワード	転送アドレス
1ワード	バイト／セクタカウント
1ワード	開始セクタ番号（キャラクタデバイスでは無視される）
2ワード	エラー（0FH）時、要求されたボリュームIDへのポインタ

ドライバはセットされたコマンドコードにしたがって、READまたはWRITEコールを実行しなければなりません。ブロックデバイスはセクタ単位でリードまたはライトし、キャラクタデバイスはバイト単位でリードまたはライトします。

I/Oが完了したらデバイスドライバはステータスワードをセットし、正常に転送されたセクタ数またはバイト数を報告しなければなりません。エラーによって転送が完了しなかった場合もこの処理を行う必要があります。エラービットとエラーコードをセットするだけでは不十分です。

ドライバはステータスワードをセットするのに加えて、セクタカウントを実際に転送されたセクタ数（またはバイト数）にセットしなければなりません。IOCTL I/Oコールについてはエラーチェックは行われません。デバイスドライバは常にリターンバイト／セクタカウントを、正常に転送された実際のバイト／セクタ数にセットしなければなりません。

ベリファイスイッチがオンの場合は、デバイスドライバはコマンドコード9（WRITE WITH VERIFY）で呼び出されます。デバイスドライバはライト動作の検証を行うことになります。

ドライバがエラーコード0FH（無効なデバイス変更）を返す場合、ドライバはASCIZ文字列（正しいボリュームID）に対するDWORDポインタを返さなければなりません。このエラーコードを返すことによって、MS-DOSに対してユーザーがディスクを再挿入するためのプロンプトを出すようトリガがかけられます。デバイスドライバはBUILD BPBファンクションの結果、ボリュームIDを読み込んでいなければなりません。

ドライバはOPENおよびCLOSEファンクションをモニタすることによって、ディスク上のオープンファイルのリファレンスカウントを保守することができます。これによってドライバはエラー0FHを返す時期を判定することができます。オープンファイルがまったくなく（リファレンスカウント＝0）、ディスクが変更されている場合はそのI/Oは成功です。これに対してオープンファイルがある場合には、0FHが存在している可能性があります。

OUTPUT UNTIL BUSYコールはプリント待ち行列専用のキャラクタデバイス上の速度を最適化します。デバイスドライバはデバイスからBUSYを返さ

れるまで、可能な限りすべてのキャラクタを出力します。環境が整備されていない場合（プリンタが準備されていない等）、デバイスドライバはこのファンクションが実行されないようにしなければなりません。デバイスドライバが要求されたバイト数より少ないバイト数を出力しても（または、0バイトを出力しても）、デバイスドライバはそれをエラーとみなさないことに注意してください。

OUTPUT UNTIL BUSYコールを使うことによって、スプーラプログラムは多くのプリンタが行メモリ（プリンタでは、1キャラクタづつ受け取って打ち出すのではなく、プリンタの1行分かまたは改行コードを受け取って1ライン分を打ち出します。この1ライン分のキャラクタを記憶しておくのが行メモリです）を利用します。

多くのプリンタはキャラクタの合計が固定されているか、または行数の制限がある受信バッファ（RAMを使用した）を持っています（受信バッファがある場合は、受信バッファから行メモリにキャラクタを渡します）。

プリンタがBUSYを返す前にバッファがfullになるか、またはキャラクタの途中でBUSYを返すことがあります。バッファはキャラクタの行をプリンタにすばやく出力することができますが、それに比べプリンタは印字するのに長い時間を必要とし、その間プリンタはBUSYの状態になります。このデバイスコールを使うことによって、バックグラウンドのスプーラプログラムがプリンタ固有のバッファの能力を使うことができます。

プリンタの場合、デバイスドライバ側で各キャラクタごとにレディ信号まで確認していると、オーバーヘッド（無駄な時間）が生じるのでBUSY信号だけのチェックだけで十分です。

次の説明はブロックデバイスドライバに適用されます。

ある種の状況においてBIOSが、BIOS I/Oパケット内の転送アドレスの〝ラップアラウンド〟を起こすと思われるような、64Kバイトの書き込み動作を実行するよう要求される場合があります。

このリクエストはMS–DOSのライトコードに追加される最適化処理によって発生するものです。これはファイル上の64Kバイトのセクタサイズ内の書き込み動作が、現在のEOFを〝通り超してしまう〟ことを意味します。このような場合、BIOSはもしそのように指定すれば〝ラップアラウンド〟する部分のライト動作を無視することが可能です。

たとえば転送アドレス×××：1で、10000Hバイト相当のセクタの書き込み動作は最後の2バイトを無視することができます。ユーザープログラムではFFFFHバイトを超えるI/Oをリクエストすることはできず、転送セグメント内で（たとえ0でも）ラップアラウンドすることはできません。したがって、この場合は最後の2バイトを無視することができます。

MS–DOSは2つのFATを保守しています。DOSが最初のFATをリードするときに問題があれば、エラーを報告する前に自動的に2番目のFATをリードしようとします。BIOSはすべての再試行に対して責任があります。

COMMAND.COMハンドラには自動的な再試行はありませんが、アプリケーションでは特定のタイプの割り込み24Hエラーに関して、それらを報告する前に

自動的再試行を行う独自の割り込み 24H ハンドラを持つものがあります。

■ NON DESTRUCTIVE READ

コマンドコード＝5

NON DESTRUCTIVE READ NO WAIT ES：BX→

13バイト	リクエストヘッダ
1バイト	デバイスからのリード

次に読み込まれた文字を返します。

ステータスワードの DONE ビットがセットされます。

キャラクタデバイスによって BUSY ビット＝0（バッファ内に文字が存在します）が返された場合、次に読み込まれる文字が返されます。この文字は入力バッファから削除されません。BUSY ビット＝1が返されたときはバッファ中に文字がありません。

■ DEVICE OPEN、DEVICE CLOSE

コマンドコード＝13、14

OPEN or CLOSE ES：BX→

13バイト	静的リクエストヘッダ

このファンクションはバージョン 3.1 以降の MS–DOS で、OPEN/CLOSE/RM のアトリビュートビットがセットされている場合のみサポートされます。デバイス上のカレントファイルの動作について、デバイスに知らせるように作られています。ブロックデバイス上ではローカルのバッファ管理に使うことができます。デバイスはリファレンスカウントを保持します。これはオープンが実行される度にカウントは1つ加算され、クローズされる度にカウントは1つ減算されます。カウントが0になった場合、デバイス上にオープンされているファイルがなくデバイスはデバイス内で使用されていて、書き込まれたバッファをすべて出力したことになります。

これはデバイスが交換可能なメディアのディスク上のメディアをユーザーが使用する場合、ユーザーの意志でディスクを交換したときにエラーを出さないようにするためです。ただし、ブロックデバイス上のこの方法は問題があります。FCB コールはファイルをクローズしないでオープンできます。したがって、メディアを交換しましたかという問いに対して〝Y〞と答えたときに、バッファ内をすべて出力せずにカウントが0になった場合、カウントをリセットしてデバイス側で BPB の作成のコールを実行するのが良い方法です。

これらのコールはほとんどキャラクタデバイスで使われます。オープンコールはデバイスが設定した文字列の先頭から送ることができます。プリンタの場合はフォント（書体）の設定、ページサイズのための特別な制御キャラクタ（文字列）

があり、これをデバイス側で処理することができます。

すべてのプロセスが標準入力、標準出力、標準エラー出力、外部装置、プリンタ（ハンドル0、1、2、3、4）にアクセスしてから、CON、AUX、PRN等の各デバイスは常にオープンされていることを知っていなければなりません。

■ REMOVABLE MEDIA

コマンドコード＝15

REMOVABLE MEDIA ES：BX→

13バイト　静的リクエストヘッダ

このファンクションはバージョン3.1以降のMS–DOSで、OPEN/CLOSE/RMのアトリビュートビットがセットされている場合のみサポートされます。このコールはIOCTLシステムコールのファンクションによってブロックデバイスでのみ使用できます。交換不可能なメディア（ハードディスクのような）、または交換可能なメディア（通常のディスクドライブ）のどちらを取り扱うかについてをユーティリティに知らせるのに使われます。

ステータスワードのBUSYのビットが返されます。BUSYが1ならばメディアは交換不可能でBUSYが0ならば交換可能です。エラービットについてはチェックされないことに注意してください。これはこのコールが失敗しないためです。

■ STATUS

コマンドコード＝6、10

STATUS ES：BX→

13バイト　リクエストヘッダ

このコールはデータが入力または出力を待っているかどうかを示す情報をMS–DOSに返します。ドライバはステータスワードとBUSYビットを、次のようにセットしなければなりません。

・キャラクタデバイスへ出力する場合

ドライバがリターン時にビット9を1にセットした場合は、MS–DOSに対してライトリクエスト（もし行われていれば）が現在のリクエストの完了を待っていることを表します。このビットが0であれば、現在のリクエストはなくライトリクエスト（もし行われていれば）はすぐに開始されます。

・バッファ付きのキャラクタデバイスから入力する場合

ドライバがリターン時にビット9を1にセットした場合は、バッファリングされているキャラクタはないことを意味し、リードリクエスト（もし行われていれば）が物理的にデバイスに行くことを意味します。0が返された場合はデバイス

バッファにキャラクタがあり、リードが拒否されないことを意味します。

0が返された場合はユーザーが何かをタイプしたことを意味します。MS-DOSはすべてのキャラクタデバイスが入力タイプアヘッドバッファを持っているものと見なします。タイプアヘッドバッファを持たないデバイスの場合は、MS-DOSが存在していないバッファへ何かが入るのを待つことがないように常にBUSY＝0を返さなければなりません。

■ FLUSH

コマンドコート＝7、11

FLUSH ES：BX→

13バイト	リクエストヘッダ

FLUSHはドライバに対し、ペンディング中のすべてのリクエストをフラッシュ（打切り）するよう指示します。このコールはキャラクタデバイスの入力キューを打ち切る目的で使用します。

デバイスドライバはFLUSHを実行してステータスワードをセットし、そしてリターンします。

■ Generic IOCTL

コマンドコード＝19

ES：BX→

13バイト	静的リクエストヘッダ
バイト	カテゴリ（メジャー）コード
バイト	ファンクション（マイナー）コード
ワード	SIの内容
ワード	DIの内容
2ワード	データバッファへのポインタ

Generic IOCTLファンクションはデバイスヘッダ内のMS-DOSバージョン3.3のアトリビュートビットを、デバイスドライバがセットしたときだけにコールされます。このコールはIOCTLシステムコールのサブファンクションによって、ブロックデバイスのみへ発せられます。

このファンクションはリクエストヘッダの静的な部分な部分に含まれる情報に加え、カテゴリ、ファンクション、の両方を含んでいます。MS-DOSはDOSコードによって実際のサービスをするデバイスコマンドであれば横取りするために、カテゴリフィールドを調べます。他のすべてのコマンドカテゴリはデバイスドライバにサービスさせるために渡されます。

Generic IOCTLファンクションは一般的に拡張されたIOCTL機能で、これまでのリードIOCTLとライトIOCTLのデバイスドライバファンクションに代

わるために作成されました。MS-DOS バージョン 2.0 の IOCTL ファンクションも、IOCTL システムコール（ファンクション 2、3、4、5）としてサポートされていますが、新しいデバイスドライバはこのファンクションを使用します。

これらのカテゴリとファンクションコードに関するより詳しい解説は、プログラマーズリファレンス VOL.1 のファンクション 440CH（一般 IOCTL：ハンドル用）と、ファンクション 440DH（一般 IOCTL：ブロックデバイス用）を参照してください。

■ DEINSTALL

コマンドコード＝20

DEINSTALL ES：BX→

13 バイト	リクエストヘッダ

DEINSTALL ファンクションはインストールされたキャラクタ系デバイスドライバをデインストール（登録されていたデバイスドライバを取り除くこと）するために、DELDRV コマンドからコールされます。このファンクションでは必要に応じて次のような処理を行う必要があります。

1. キャラクタ系デバイスドライバが MS-DOS のシステム環境を変更している場合には、変更したシステム環境をもとに戻す必要があります。

2. デバイスドライバが ROM BIOS あるいは直接ハードウェアをアクセスし、そのモードなどを変更している場合には、変更したものをもとに戻す必要があります。

注意　この DEINSTALL ファンクションは、INIT ファンクション（コマンドコード 0）で動的な登録、削除機能をサポートしていることを宣言している場合のみコールされます（宣言の方法については、INIT ファンクションを参照してください）。

■ Get/Set Logical Drive Map

コマンドコード＝23（取得）または 24（設定）

Get/Set Logical Drive Map ES：BX→

13 バイト	静的リクエストヘッダ
バイト	入力（ユニットコード）
バイト	出力（最後のデバイス参照）
バイト	コマンドコード
ワード	ステータス
2 ワード	予約

Get/Set Logical Drive Map ファンクションはデバイスヘッダ内の MS–DOS バージョン 3.3 のアトリビュートビットを、デバイスドライバがセットしたときだけにコールされます。このコールは IOCTL システムコールのサブファンクションによって、ブロックデバイスのみへ発せられます。マップされた論理ドライブはヘッダのユニットコードフィールドでデバイスドライバへ渡されます。デバイスドライバは要求によってマップされた物理ドライブのオーナーである現在の論理ドライブを返します。

論理ドライブがマップされた物理ドライブを現在所有しているかどうかを検出するために、プログラムはファンクション 440EH または 440FH（論理ドライブマップの取得／設定）をコールした後で、ユニットコードフィールドが変更されていないことを検証（ベリファイ）する必要があります。

1.8　メディアディスクリプタバイト

メディアディスクリプタバイトは、MS–DOS に対して異なるタイプのメディアが存在することを通知するために使用されます。

1つのドライブで何種類ものディスクをサポートする場合、何種類もの BPB が存在します。このとき複数の BPB のうち現在 MS–DOS が使用しているのは、どの BPB であるかを知るために BPB の中にメディアディスクリプタバイトと呼ばれる1バイトのある値をセットします。この値は 00H〜FFH の範囲の任意の値をとることができます。

このバイトは FAT ID バイトと同じである必要はありません。FAT の最初のバイトである FAT ID バイトは MS–DOS バージョン 2.0 以前では、異なるタイプのディスクメディアを区別する目的で使用されていましたが、バージョン 2.0 以降のディスクデバイスドライバでも同様に使用することができます。ただし FAT ID バイトは NON FAT ID ビットがセットされない状態で、かつブロックデバイスドライバでのみ意味を持ちます。

メディアディスクリプタバイトまたは FAT ID バイトの値は、MS–DOS に対しては何の意味も持ちません。これらのバイトは単にメディアの判定を容易に行えるようにデバイスドライバに渡されるものです。

注意　BUILD BPB コールを行ったときに、新しい BPB で返されたメディアバイトが以前のメディアバイトと同じ場合には、MS–DOS はそのデバイスに関して内部構造を再構成しません。MS–DOS は物理的ディスクが変更された場合にも、フォーマットが変更されていないものとしてディスクを取り扱います。したがって、それぞれの BPB は唯一のメディアディスクリプタバイトを持っていなければなりません。

1.9 メディアディスクリプタテーブル

MS–DOSのファイルシステムはファイルアロケーションテーブル（FAT）というポインタ（各クラスタまたはアロケーションユニットに対する）のリンクされたリストを使用します。未使用のクラスタは0で表され、エンドオブファイルはFFFH（16ビットのFATエントリを持つユニットではFFFFH）で表されます。通常、有効なエントリは0エントリを指しませんが、もし指した場合は最初のFATエントリ（0エントリによって指される）は予約されエンドオブチェインにセットされます。その結果、いくつかのエンドオブチェイン値が定義され（(F) FF8〜（F）FFF)、これらは異なるメディアのタイプを区別するために使用されます。

ここでの最善の方法はブートセクタに完全なメディアディスクリプタテーブルを書き、それをメディアの識別に使用することです。NON FAT IDビットをセットしないドライバを持つシステムのMS–DOSのバージョンに互換性を保証するために、FORMATのプロセス中にFAT IDバイトを書き込むことも必要です。

今後、さまざまなディスクフォーマットをサポートするための柔軟性を高めるためには、特定の種類のメディアについてのBPBに関する情報をブートセクタに保存しておくことが望ましいと言えます。このようなブートセクタのフォーマットを次に示します。

	サイズ	内容
	3バイト	ブートコードへのnear jump
	8バイト	メーカー名およびバージョン
BPB	1ワード	1セクタあたりのバイト数
BPB	1バイト	1アロケーションユニットあたりのセクタ数
BPB	1ワード	予備のセクタ数
BPB	1バイト	FATの数
BPB	1ワード	ルートディレクトリエントリの数
BPB	1ワード	論理イメージ内のセクタ数
BPB	1バイト	メディアディスクリプタ
BPB	1ワード	1FATセクタ数
	1ワード	1トラックあたりのセクタ数
	1ワード	ヘッド数
	1ワード	隠れたセクタの数

最後の3つのワード（トラック当りのセクタ数、ヘッダ数、隠れたセクタの数）は、MS–DOSによっては使用されませんが、デバイスドライバで使用することができます。これらはデバイスドライバがメディアを理解するための補助手段です。それぞれ次のような意味を持ちます。

・トラック当りのセクタ数、ヘッダ数
論理的レイアウトが同じで物理的レイアウトが異なるメディア（例：40 トラック両面と、80 トラック片面）をサポートする場合に使用することができます。

・トラック当りのセクタ数
デバイスドライバに対し物理的なディスク上に、論理的ディスクフォーマットがどのようにレイアウトされているかを指示します。

・隠れたセクタの数
ドライブのパーティション（区画化）スキーマをサポートするため使用することができます。

NON FAT ID フォーマットドライバによるメディア判定には、次のプロシージャを推奨します。

1. ドライブのブートセクタを、DWORD 転送アドレスによって指定される 1 セクタのスクラッチスペースに読み込みます。

2. ブートセクタの最初のバイトが E9H または EBH (3 バイト NEAR または 2 バイト SHORT JUMP の最初のバイト）か、あるいは EBH（後に NOP が続く 2 バイト JUMP の最初のバイト）のいずれかであることを判定します。もしそうならば BPB はオフセットを基点としてかれ、それに対するポインタを返します。

3. ブートセクタに BPB テーブルがない場合には、それは MS-DOS のバージョン 2.0 以前でフォーマットされたディスクであり、FAT ID バイトを用いてメディアを判定します。

ドライバはオプションとして FAT の最初のセクタを 1 セクタのスクラッチ領域に読み込み、最初のバイトを読むことによってメディアタイプを判定することができます。この判定はシステムによって読まれるべきディスク上で使用されている FAT ID バイトに基づくものとします。ハードコーデッド BPB に対するポインタを返します。

1.10　クロックデバイス

よく用いられる追加ボードにリアルタイムクロックボードがあります。時刻および日付を得るためにこのボードをシステム内に統合できるようにする（アトリビュートワードによって決定される）クロックデバイスという特殊な装置があります。クロックデバイスは他のすべてのキャラクタデバイスと同様にファンクションを定義し、それを実行します。ファンクションの大部分は、〝DONE ビットをセットし、エラービットをリセットしてリターンする〟というものです。このデバイスに対してリード／ライトが行われると、6 バイトの値が転送されます。先頭の 2 バイトは 1980 年 1 月 1 日から数えた日数のワード、3 バイト目は分、4 バイト目は時、5 バイト目は 1/100 秒、6 バイト目は秒を示します。クロックデバイスリードによって日付および時刻を取得し、ライトによって日付および時刻をセットします。

1.11 デバイスコールの分析

MS–DOSが、ライトリクエストを実行するためにブロックデバイスドライバを呼び出したときの動作を具体的に説明します。

1. MS–DOSはメモリの予約された領域に、リクエストパケット（リクエストヘッダ+パラメータ）を書き込みます。

2. MS–DOSはブロックデバイスドライバのストラテジエントリポイントを呼び出します。

3. デバイスドライバはESおよびBXレジスタをセーブし（ES：BXはリクエストパケットを指します）、そしてFARリターンを行います。

4. MS–DOSは割り込みエントリポイントを呼び出します。

5. デバイスドライバはリクエストパケットへのポインタを取得しコマンドコード（オフセット2）を読んで、これがライトリクエストであることを判定します。デバイスドライバはこのコマンドコードをディスパッチテーブルへの索引に変換し、制御はディスクライトルーチンに渡されます。

6. デバイスドライバはユニットコード（オフセット1）を読んで、どのディスクドライブでライト動作を行うかを判定します。

7. コマンドはディスクライトなのでデバイスドライバはリクエストパケットから、転送アドレス（オフセット14）、セクタカウント（オフセット18）、およびスタートセクタ（オフセット20）を入手しなければなりません。

8. デバイスドライバは最初の論理セクタ番号をトラック、ヘッド、およびセクタ番号に変換します。

9. デバイスドライバはユニットコード（このデバイスドライバによって定義されるサブユニット）で定義されるドライブの指定された開始セクタから開始して、指定された数のセクタを書き込み、リクエストパケットで示される転送アドレスからデータを転送します。この中にはディスクコントローラに対する複数のライトコマンドが含まれていてもかまいません。

10. 転送が完了したら、デバイスドライバはステータスワード（リクエストパケットのオフセット3）の中のDONEビットをセットすることによって、MS–DOSに対してリクエストのステータスを報告しなければなりません。リクエストパケットのセクタカウント領域に、実際に転送されたセクタ数を報告します。

11. エラーが発生した場合は、ドライバはステータスワード内のDONEビットおよびエラービットをセットし、ステータスワードの下半分にエラーコードを書き込みます。実際に転送されたセクタ数をリクエストパケットに書き込まなければなりません。ステータスワードのエラービットをセット

するだけでは不十分です。

12. デバイスドライバはMS–DOSへFARリターンを行います。

デバイスドライバはMS–DOSの状態を保存しなければなりません。これはすべてのレジスタ（フラグも含めて）を保存するということになります。このとき方向フラグと割り込みイネーブルビットは重要になります。デバイスドライバの割り込みエントリポイントが呼び出される時点で、MS–DOSは内部スタックに約40〜50バイトのあきがあります。デバイスドライバが拡張的なスタック動作を使用する場合は、ローカルスタックに切り換えなければなりません。

1.12　デバイスドライバ例

次にブロックデバイスドライバおよびキャラクタデバイスドライバのプログラム例を掲載します。このプログラム例は、そのままではアセンブルして動作可能なものではありません。

■ ブロックデバイスドライバ

```
;
;       デバイスドライバ          (ブロックデバイス)
;
; このプログラムは、ブロックデバイスの例です
CODE            SEGMENT
                ASSUME CS:CODE,DS:CODE,ES:CODE,SS:CODE
;**********************************************************************
;
;   デバイスヘッダ
;
;   デバイスヘッダは、次の形式です
;
;     次のデバイスヘッダへのポインタ           ダブルワード
;     最後のデバイスヘッダの場合は、-1 とする
;
;     アトリビュート                          ワード
;       15 ビット目  1    =  キャラクタデバイス
;                    0    =  ブロックデバイス
;       15 ビット目が 1 (キャラクタデバイス) の場合
;         0 ビット目  1 ならば、標準入力用デバイス
;         1 ビット目  1 ならば、標準出力用デバイス
;         2 ビット目  1 ならば、ヌルデバイス (DOS で使用)
;         3 ビット目  1 ならば、クロックデバイス
;         4 ビット目-12 ビット目
```

```
;           すべて 0 にする
;         13 ビット目 (ブロックデバイスのみ)
;           0 ならば、IBM フォーマット
;         14 ビット目
;           1 ならば、コントロール文字列の処理可能
;
;     ストラテジルーチンへのエントリポイント          ワード
;
;     割り込みルーチンへのエントリポイント            ワード
;
;     名前                                           8 バイト
;
;     キャラクタデバイスの場合は、8 文字でデバイスの名前を指定する
;     ブロックデバイスの場合は、先頭の 1 バイトでユニット数を指定する
;
DSKDEV  LABEL   WOR
        DD      -1                  ; 最後のデバイス
        DW      0000H               ; ブロック、IBM フォーマット
        DW      STRATEGY            ; ストラテジエントリポイント
        DW      DSK_INT             ; 割り込みエントリポイント
DSKNUM  DB      2                   ; ユニット数

; このテーブルは、コマンドを割振るためのものです

DSKTBL  LABEL   WOR
        DW      DSK_INIT            ;INIT
        DW      MEDIA_CHK           ;MEDIA CHECK
        DW      GET_BPB             ;BUILD BPB
        DW      CMD_ERR             ;IOCTL INPUT
        DW      DSK_READ            ;INPUT
        DW      EXIT                ;NON-DESTRUCTIVE INPUT NO WAIT
        DW      EXIT                ;INPUT STATUS
        DW      EXIT                ;INPUT FLUSH
        DW      DSK_WRT             ;OUTPUT
        DW      DSK_WRTV            ;OUTPUT & VERIFY
        DW      EXIT                ;OUTPUT STATUS
        DW      EXIT                ;OUTPUT FLUSH
        DW      EXIT                ;IOCTL OUTPUT

        PAGE
;***********************************************************************
;
```

```
;		ストラテジルーチン
;
PTRSAV		DD	0			; リクエストヘッダアドレス退避用

STRATP		PROC	FAR
STRATEGY:
		MOV	WORD PTR CS:[PTRSAV],BX
						; リクエストヘッダアドレスの退避
		MOV	WORD PTR CS:[PTRSAV+2],ES
		RET
STRATP		ENDP

		PAGE
;***********************************************************************
;
;		割り込みルーチン
;
;		リクエストヘッダの定義

REQ_HEAD	STRUC
CMDLEN		DB	?			; コマンド長
UNIT		DB	?			; ユニットコード
CMD		DB	?			; コマンドコード
STATUS		DW	?			; ステータス
		DB	8 DUP(?)
MEDIA		DB	?			; メディアディスクリプタバイト
TRANS		DD	?			; 転送アドレス
COUNT		DW	?			; セクタカウント
START		DW	?			; 開始セクタ番号
REQ_HEAD	ENDS
;
;		割り込みルーチン
;
DSK_INT:
		PUSH	SI			; レジスタの退避
		PUSH	AX
		PUSH	CX
		PUSH	DX
		PUSH	DI
		PUSH	BP
		PUSH	DS
		PUSH	ES
```

```
                PUSH    BX
                LDS     BX,CS:[PTRSAV]  ; リクエストヘッダアドレスの取得

                MOV     AL,[BX.UNIT]    ; ユニットコードの取得
                MOV     AH,[BX.MEDIA]   ; メディアディスクリプタバイトの取得
                MOV     CX,[BX.COUNT]   ; セクタカウントの取得
                MOV     DX,[BX.START]   ; 開始セクタ番号の取得
                XCHG    AX,DI           ; レジスタ AX の退避
                MOV     AL,[BX.CMD]     ; コマンドコードの取得
                CMP     AL,11
                JA      CMD_ERR         ; コマンドエラー
                XOR     AH,AH
                SHL     AX,1
                MOV     SI,OFFSET DSKTBL
                ADD     SI,AX
                XCHG    AX,DI           ; レジスタ AX の復帰
                LES     DI,[BX.TRANS]   ; 転送アドレスの取得
                PUSH    CS
                POP     DS              ;DS  ←  CS
                JMP     [SI]            ; コマンド処理ルーチンへ

                PAGE
;*************************************************************************
;
;       出口
;
CMD_ERR:                                ; コマンドエラー
                MOV     AL,3            ; エラーコードのセット
ERR_EXIT:                               ; エラーリターン
                MOV     AH,10000001B    ; エラービット、DONE ビットのセット
                JMP     SHORT EXIT1

EXITP           PROC    FAR
EXIT:                                   ; ノーエラーリターン
                MOV     AH,00000001B    ;DONE ビットのセット
EXIT1:
                LDS     BX,CS:[PTRSAV]  ; リクエストヘッダアドレスの取得
                MOV     [BX.STATUS],AX  ; ステータスのセット

                POP     BX              ; レジスタの復帰
                POP     ES
                POP     DS
```

```
		POP	BP
		POP	DI
		POP	DX
		POP	CX
		POP	AX
		POP	SI
		RET
EXITP		ENDP

		PAGE
;*************************************************************************
;
;	コマンド処理ルーチン
;
;*************************************************************************
;
MEDIA_CHK:					;MEDIA CHECK
		LDS	BX,[PTRSAV]	;リクエストヘッダアドレスの取得
		MOV	BYTE PTR [BX.TRANS],0
					;変更されたかわからない
		JMP	EXIT
;*************************************************************************
;
GET_BPB:					;BUILD BPB
		MOV	AH,ES:[DI]	;FAT ID バイトの取得
		MOV	SI,OFFSET DSKBPB
					;9 セクタ／トラックをセット
		CMP	AH,0F9H		;FAT ID は 9 セクタ／トラックか
		JE	DSK_9
		MOV	SI,OFFSET DSKBPB1
					;8 セクタ／トラックをセット
DSK_9:
		LDS	BX,[PTRSAV]	;リクエストヘッダアドレスを取得
		MOV	[BX.MEDIA],AH	;メディアディスクリプタをセット
		MOV	[BX.COUNT],SI	;BPB へのポインタをセット
		MOV	[BX.COUNT+2],CS
		JMP	EXIT
;
;
DSK_READ:					;INPUT
		MOV	RW_FLG,0	;リードをセット
		MOV	VRFY_FLG, 0
```

```
RW_COMMON:
                JCXZ    NO_IO           ; セクタカウントが 0 なら、何もしない
                CALL    RW_EXE
                PUSHF                   ; キャリーフラグを退避
                LDS     BX,[PTRSAV]     ; リクエストヘッダアドレスを取得
                SUB     [BX.COUNT],CX   ; 転送したセクタ数のセット
                POPF                    ; キャリーフラグの復帰
                                        ;CF  =  0   ノーエラー
                                        ;    =  1   エラー
                JC      ERR_EXIT
NO_IO:
                JMP     EXIT
;
DSK_WRT:                                ;OUTPUT
                MOV     RW_FLG,1        ; ライトをセット
                MOV     VRFY_FLG,0      ; ベリファイをオフ
                JMP     RW_COMMON
;
DSK_WRTV:                               ;OUTPUT & VERIFY
                MOV     RW_FLG,1        ; ライトをセット
                MOV     VRFY_FLG, 1     ; ベリファイをオン
                JMP     RW_COMMON

                PAGE
;***********************************************************************
;
;       入力：
;       AL ＝ユニットコード
;       AH ＝メディアディスクリプタ
;       CX ＝セクタカウント
;       DX ＝開始セクタ番号
;       ES：DI ＝転送アドレス
;       出力：
;       成功：キャリーフラグ＝ 0
;       失敗：              ＝ 1
;       AL ＝エラーコード
;       CX ＝転送されなかったセクタカウント
RW_EXE:
                MOV     UNIT_NO,AL      ; ユニットコードの退避
                MOV     SEC_CNT,CX      ; セクタカウントの退避
                CALL    SET_UP          ; 開始セクタ番号 → 物理セクタ番号
LOOP_RW:
```

```
            CALL    RW
            JC      RW_ERR
            CALL    SET_UP1         ; 次のセクタにセット
            DEC     SEC_CNT
            JNZ     LOOP_RW
RW_ERR:
            MOV     CX,SEC_CNT      ; 残りのセクタ数
            RET
;***********************************************************************
;
;       入力：
;       DX ＝開始セクタ番号
;       出力：
;       BX ＝転送する長さ（バイト）
;       CH ＝セクタ長
;       CL ＝シリンダ番号
;       DH ＝ヘッド番号
;       DL ＝セクタ番号
SET_UP:
            MOV     BL,9            ;9 セクタ／トラック
            CMP     AH,0F9H         ; メディアディスクリプタバイトの
                                    ; チェック
            JE      DSK9_01
            MOV     BL,8            ;8 セクタ／トラック
DSK9_01:
            MOV     SEC_TRK,BL
            MOV     AX,DX           ;AX  ←  開始セクタ番号
            DIV     BL
            INC     AH
            MOV     DL,AH           ; セクタ番号のセット
            XOR     DH,DH
            SHR     AL,1
            RCL     DH,1            ; ヘッド番号のセット
            MOV     CL,AL           ; シリンダ番号のセット
            MOV     BX,512          ; 転送長のセット
            MOV     CH,02H          ; セクタ長のセット
                                    ;0：128 バイト  1：256 バイト
                                    ;2：512 バイト  3：1024 バイト
            RET
;
SET_UP1:
            INC     DL              ; 次のセクタ
```

```
                CMP     DL,SEC_TRK      ; 次のトラックか
                JBE     NOT_NXT
                MOV     DL,1            ; セクタ#1 をセット
                XOR     DH,1            ; ヘッドを反転
                JNZ     SAME_CYL
                INC     CL              ; 次のシリンダへ
SAME_CYL:
NOT_NXT:
                ADD     DI,BX           ; バッファのアップデート
                RET
;
RW:
                CMP     RW_FLG,0        ; リードか
                JNE     WRT
;
;       ここで、リードをする
;       エラーが起こったら、キャリーフラグをセット
;
                JC      ERROR_SET       ; エラーならジャンプ
                RET
WRT:
;
;       ここで、ライトをする
;       エラーが起こったら、キャリーフラグをセット
;
                JC      ERROR_SET       ; エラーならジャンプ
                CMP     VERY_FLG,0      ; ベリファイオフか
                JE      NOT_VRFY
;
;       ここで、ベリファイをする
;       エラーが起こったら、キャリーフラグをセット
;
                JC      ERROR_SET       ; エラーならジャンプ
NOT_VRFY:
                RET
ERROR_SET:
;
;       ここで、レジスタ AL にエラーコードをいれる
;
                STC
                RET
;
```

```
;           データ
;
RW_FLG      DB      ?               ;0：リード
                                    ;1：ライト
VRFY_FLG    DB      ?               ;0：ベリファイオフ
                                    ;1：ベリファイオン
UNIT_NO     DB      ?               ;ユニットコード
SEC_TRK     DB      ?               ;セクタ数／トラック
SEC_CNT     DW      ?               ;セクタカウント
            PAGE
;*************************************************************************
;
;             BPB の定義
;
DSKBPB:                             ;5 インチ 2DD（9 セクタ／トラック）
            DW      512             ;物理セクタ長（バイト）
            DB      1               ;セクタ数／アロケーションユニット
            DW      1               ;リザーブセクタ数
            DB      2               ;FAT のコピー数
            DW      112             ;ディレクトリエントリ数
            DW      9*2*80          ;セクタ数／ボリューム
            DB      0F9H            ;メディアディスクリプタバイト
            DW      3               ;セクタ数／FAT
DSKBPB1:                            ;5 インチ 2DD（8 セクタ／トラック）
            DW      512
            DB      1
            DW      1
            DB      2
            DW      112
            DW      8*2*80
            DB      0FBH
            DW      2

INIT_TBL:
            DW      OFFSET DSKBPB   ;ユニット  #0
            DW      OFFSET DSKBPB   ;ユニット  #1
;*************************************************************************
;
DSK_INIT:                           ;INIT
            MOV     AL,DSKNUM       ;ユニット数の取得
            PUSH    DS              ;レジスタ DS の退避
            LDS     BX,[PTRSAV]     ;リクエストヘッダアドレスの
```

```
                                        ; 取得
                MOV     [BX.MEDIA],AL   ; ユニット数のセット
                MOV     WORD PTR[BX.TRANS],OFFSET DSK_INIT
                MOV     WORD PTR[BX.TRANS+2],CS
                MOV     WORD PTR[BX.COUNT+2],CS
                                        ; ブレークアドレスのセット
                MOV     WORD PTR[BX.COUNT],OFFSET INIT_TBL
                                        ;BPB 配列アドレスのセット
                POP     DS              ; レジスタ DS の復帰
                JMP     EXIT

                CODE    ENDS
                        END
```

■ キャラクタデバイスドライバ

次は、キャラクタデバイスドライバのプログラム例です。

```
;       キャラクタデバイスドライバのプログラム例

CODE            SEGMENT BYTE
                ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
;           EQU
CR              EQU     13                  ; キャリッジリターンコード
BACKSP          EQU     8                   ; バックスペースコード
_ESC            EQU     1BH                 ; エスケープコード
BASE            EQU     0A000H              ;VRAM セグメント
;-----------------------------------------------------------
;
;       CON     コンソールデバイスドライバの例
;
CONDEV          DW      -1, -1
                DW      1000000000000011B   ;CON_IN と CON_OUT のサポート
                DW      STRATEGY
                DW      ENTRY
                DB      'CON'
;-----------------------------------------------------------
;
;       コマンドジャンプテーブル
;
CONTBL          DW      CON_INIT            ; 初期設定
                DW      EXIT                ; 媒体検査
                DW      EXIT                ;BPB 作成（ブロックのみ）
```

```
        DW      CMDERR              ;IOCTL 入力
        DW      CON_READ            ; 入力（入力待ち）
        DW      CON_RDND            ; 連続非破壊読み込み
        DW      EXIT                ; 入力状況
        DW      CON_FLSH            ; 入力要求終了
        DW      CON_WRIT            ; 出力（書き込み）
        DW      CON_WRIT            ; 出力（書き込み）
        DW      EXIT                ; 出力状況
        DW      EXIT                ; 出力要求終了
        DW      EXIT                ;IOCTL 出力

CMDTABL DB      'A'
        DW      CUU                 ;CURSOR UP
        DB      'B'
        DW      CUD                 ;CURSOR DOWN
        DB      'C'
        DW      CUF                 ;CURSOR FORWARD
        DB      'D'
        DW      CUB                 ;CURSOR BACK
        DB      'H'
        DW      CUH                 ;CURSOR POSITION
        DB      'Y'
        DW      CUP                 ;CURSOR POSITION
        DB      'j'
        DW      PSCP                ;SAVE CURSOR POSITION
        DB      'k'
        DW      PRCP                ;RESTORE CURSOR POSITION
        DB      00
        PAGE
;-----------------------------------------------------------
;
;       割り込みルーチン
;
CMDLEN          EQU     0           ; コマンド長
UNIT            EQU     1           ; ユニットコード
CMD             EQU     2           ; コマンドコード
STATUS          EQU     3           ; ステータス
MEDIA           EQU     13          ;
TRANS           EQU     14          ; 転送アドレス
COUNT           EQU     18
START           EQU     20
```

```
PTRSAV          DD          0

STRATP          PROC        FAR
;
;           ストラテジ
;
STRATEGY:
                MOV         WORD PTR CS:[PTRSAV],BX
                                                    ; リクエストヘッダのオフセット
                                                    ; を保管
                MOV         WORD PTR CS:[PTRSAV+2],EX
                                                    ; セグメントを保管
                RET

STRATP  ENDP
;
;           共通入口点
;
ENTRY:
                PUSH        SI
                PUSH        AX
                PUSH        CX
                PUSH        DX
                PUSH        DI
                PUSH        BP
                PUSH        DS
                PUSH        ES
                PUSH        BX

                LDS         BX,CS:[PTRSAV]          ; リクエストヘッダアドレス
                                                    ; の取得
                MOV         CX,WORD PTR[BX.COUNT]
                                                    ; バイトカウントの取得
                MOV         AL,BYTE PTR[BX.CMD]     ; コマンドコードの取得
                CBW
                MOV         SI,OFFSET CONTBL
                ADD         SI,AX
                ADD         SI,AX
                CMP         AL,12
                JA          CMDERR
                LES         DI,[BX.TRANS]
                PUSH        CS
```

```
                POP     DS

                JMP     WORD PTR[SI]

                PAGE
;--------------------------------------------------------------
;
;       出口
;
BUS_EXIT:
                MOV     AH,10000011B            ; ビジービット、
                                                ;DONEビットをセット
                JMP     SHORT ERR1

CMDERR:
                MOV     AL,3

ERR_EXIT:
                MOV     AL,00000001B            ; エラービット、
                                                ;DONEビットをセット
                JMP     SHORT ERR1

EXITP           PROC    FAR

EXIT:
                MOV     AH,00000001B            ; 処理済みビットをセット
ERR1:
                LDS     BX,CS:[PTRSAV]
                MOV     WORD PTR[BX.STATUS],AX

                POP     BX
                POP     ES
                POP     DS
                POP     BP
                POP     DI
                POP     DX
                POP     CX
                POP     AX
                POP     SI
                RET
EXITP           ENDP
;--------------------------------------------------------------
```

```
;
;       コンソール出力サブルーチン
;

STATE           DW      S1
MAXCOL          DB      79
COL             DB      0                       ; カラム
ROW             DB      0                       ; ライン
SAVCR           DW      0                       ; カーソル位置セーブ用

ATTR            DB      11100001B               ; 文字アトリビュート
;
;       VRAM アドレステーブル
;
LINE_TABLE      LABEL   WORD
                DW      0000H                   ;0-LINE
                DW      00A0H                   ;1
                DW      0140H                   ;2
                DW      01E0H                   ;3
                DW      0280H                   ;4
                DW      0320H                   ;5
                DW      03C0H                   ;6
                DW      0460H                   ;7
                DW      0500H                   ;8
                DW      05A0H                   ;9
                DW      0640H                   ;10
                DW      06E0H                   ;11
                DW      0780H                   ;12
                DW      0820H                   ;13
                DW      08C0H                   ;14
                DW      0960H                   ;15
                DW      0A00H                   ;16
                DW      0AA0H                   ;17
                DW      0B40H                   ;18
                DW      0BE0H                   ;19
                DW      0C80H                   ;20
                DW      0D20H                   ;21
                DW      0DC0H                   ;22
                DW      0E60H                   ;23
                DW      0F00H                   ;24

CHROUT:
```

```
                CMP     AL,13
                JNZ     TRYLF
                MOV     [COL],0                 ; キャリッジリターン処理
                                                ; カラムを 0 にセット
                JMP     SHORT SETIT
TRYLF:
                CMP     AL,10                   ;LF か?
                JZ      LF
                CMP     AL,7
                JNE     TRYBACK
                                                ; ブザー処理
                MOV     AL,06H
                OUT     37H,AL                  ; ブザー ON
                MOV     CX,6000H
BEL1:
                PUSH    AX
                POP     AX
                LOOP    BEL1
                MOV     AL,07H
                OUT     37H,AL                  ; ブザー OFF
RET5:           RET

TRYBACK:
                CMP     AL,8                    ;BS か?
                JNZ     OUTCHR
                CMP     [COL],0                 ; バックスペース処理
                JZ      RET5
                DEC     [COL]
                JMP     SHORT SETIT
;
;   文字の表示
;
OUTCHR:
                XOR     AH,AH
                MOV     DH,[ROW]
                MOV     DL,[COL]
                CALL    LCCONV                  ; ライン、カラムから
                                                ;VARM アドレスを得る
                MOV     DI,BX
                MOV     DX,[BASE]
                MOV     ES,DX
                MOV     EX:[DI],AX              ; 文字のセット
```

```
        MOV     AL,[ATTR]
        ADD     DI,2000H
        STOSW                           ;アトリビュートのセット
        INC     [COL]
        MOV     AL,[COL]
        CMP     AL,[MAXCOL]
        JBE     SETIT
        MOV     [COL],0
LF:                                     ;ラインフィード処理
        INC     [ROW]
        CMP     [ROW],24                ;現在行に 1 加えて
        JB      SETIT                   ;24 以上になったら
                                        ;スクロールアップする
        MOV     [ROW],23
        CALL    SCROLL
SETIT:                                  ;カーソル表示処理
        MOV     DH,ROW
        MOV     DL,COL
        CALL    LCCONV
        SHR     BX,1
        CLI
SETIT1:
        IN      AL,60H
        TEST    AL,04H
        JZ      SETIT1
        MOV     AL,49H
        OUT     62H,AL
        MOV     AL,BL
        OUT     60H,AL
        MOV     AL,BH
        OUT     60H,AL
        STI
        RET
SCROLL:                                 ;スクロールアップ処理
        CLD
        PUSH    DS
        MOV     AX,[BASE]
        MOV     ES,AX
        MOV     DS,AX
        XOR     DI,DI
        MOV     SI,80*2
        MOV     CX,23*80
```

```
                REP     MOVSW                   ; コードデータ移送
                MOV     DI,2000H
                MOV     SI,2000H+80*2
                MOV     CX,23*80
                REP     MOVSW                   ; アトリビュートデータ移送
                MOV     DH,23
                XOR     DL,DL
                CALL    LCCONV
                MOV     DI,BX
                MOV     AL, ' '
                MOV     CX,80
                REP     STOSW                   ; 最終行スペースクリア
                MOV     AL,[ATTR]
                MOV     CX,80
                MOV     DI,BX
                ADD     DI,2000H
                REP     STOSW
                POP     DS
                RET
LCCONV:                                         ; ライン、カラムから
                                                ;VRAM アドレスへの変換
                XOR     BX,BX
                MOV     BL,DH
                SHL     BX,1
                ADD     BX,OFFSET LINE_TABLE
                MOV     BX,CS:[BX]
                XOR     DH,DH
                SHL     DL,1
                ADD     BX,DX
                RET
;--------------------------------------------------
;
;       コンソール入力ルーチン
;
CON_READ:
                JCXZ    CON_EXIT
CON_LOOP:
                PUSH    CX
                CALL    CHRIN                   ;AL に入力文字を得る
                POP     CX
                STOSB
                LOOP    CON_LOOP
```

```
CON_EXIT:
                JMP     EXIT
;----------------------------------------------------------
;
;       コンソール入力サブルーチン
;
CHRIN:
                MOV     AH,1                    ; キーボード入力チェック
                INT     18H
                CMP     BH,1
                JNE     CHRIN
                MOV     AH,0                    ; 入力文字の引き取り
                INT     18H
                RET
;----------------------------------------------------------
;
;       連続非破壊読み込み
;
CON_RDND:
                MOV     AH,1                    ; キーボード入力チェック
                INT     18H
                CMP     BH,1
                JNE     CONBUS                  ; 入力なし
RDEXIT:
                LDS     BX,[PTRSAV]
                MOV     [BX.MEDIA],AL
EXVEC:          JMP     EXIT
CONBUS:         JMP     BUS_EXIT                ; 入力なし（ビジー）
;----------------------------------------------------------
;
;       取り消しコール
;
CON_FLSH:
                MOV     AH,3
                INT     18H                     ; キーボード初期化
                JMP     EXVEC
;----------------------------------------------------------
;
;       出力（書き込み）
;
CON_WRIT:
                JCXZ    EXVEC
```

```
CON_LP:
            MOV     AL,ES:[DI]          ;出力文字 → AL
            INC     DI
            CALL    OUTC
            LOOP    CON_LP
            JMP     EXVEC
OUTC:
            PUSH    AX
            PUSH    CX
            PUSH    DX
            PUSH    SI
            PUSH    DI
            PUSH    ES
            PUSH    BP
            CALL    VIDEO
            POP     BP
            POP     ES
            POP     DI
            POP     SI
            POP     DX
            POP     CX
            POP     AX
            RET
;----------------------------------------------------------
;
;
VIDEO:
            MOV     SI,OFFSET STATE
            JMP     [SI]
S1:
            CMP     AL,ESC              ;エスケープシーケンス?
            JNE     S1B
            MOV     WORD PTR [SI],OFFSET S2
                                        ;ESCシーケンス処理ルーチン
                                        ;のアドレス
            RET

S1B:        CALL    CHROUT
S1A:        MOV     WORD PTR [STATE], OFFSET S1
            RET
;
;  ESCシーケンス解析ルーチン
```

```
;
S2:
            MOV     BX,OFFSET CMDTABL-3
S7A:
            ADD     BX,3
            CMP     BYTE PTR [BX],0
            JZ      S1A
            CMP     [BX],AL
            JNE     S7A
            JMP     WORD PTR [BX+1]      ；該当サブルーチンへ飛ぶ

MOVCUR:
            CMP     [BX],AH
            JE      SETCUR
            ADD     [BX],AL
SETCUR:
            CALL    SETIT
            JMP     S1A
CUP:                                     ；カーソルアドレッシング
            MOV     WORD PTR [SI], OFFSET CUP1
            RET
CUP1:
            SUB     AL,32
            MOV     [ROW],AL
            MOV     WORD PTR [SI], OFFSET CUP2
            RET
CUP2:
            SUB     AL,32
            MOV     [COL],AL
            JMP     SETCUR

SM:
            MOV     WORD PTR [SI], OFFSET S1A
            RET

CUH:                                     ；カーソルをホーム位置へ移動する
            MOV     WORD PTR COL,0
            JMP     SETCUR

CUF:                                     ；カーソルを右へ移動する
            MOV     AH,MAXCOL
            MOV     AL,1
```

```
CUF1:
            MOV     BX,OFFSET COL
            JMP     MOVCUR

CUB:                                    ; カーソルを左へ移動する
            MOV     AX,00FFH
            JMP     CUF1

CUU:                                    ; カーソルを上へ移動する
            MOV     AX,00FFH
CUU1:
            MOV     BX,OFFSET ROW
            JMP     MOVCUR

CUD:                                    ; カーソルを下へ移動する
            MOV     AX,23*256+1
            JMP     CUU1

PSCP:                                   ; 現在のライン、カラムを退避
            MOV     AX,WORD PTR COL
            MOV     SAVCR,AX
            JMP     SETCUR

PRCP:                                   ; 退避したライン、カラムを復帰
            MOV     AX,SAVCR
            MOV     WORD PTR COL,AX
            JMP     SETCUR
;----------------------------------------------------------
;
;       初期設定
;
CON_INIT:
            PUSH    DS
            LDS     BX,[PTRSAV]
            MOV     WORD PTR[BX.TRANS],OFFSET CON_INIT
            MOV     WORD PTR[BX.TRANS+2],CS
            POP     DS
            JMP     EXIT

CODE    ENDS
        END
```

第2章

日本語処理

2.1 イントロダクション

MS–DOSにはスムーズに日本語（漢字やひらがななどの2バイトコード文字）を入力するための拡張機能が用意されています。この拡張機能はデバイスドライバの形で提供され、ユーザーがプログラム内で利用することができます。

日本語入力用デバイスドライバには2種類あり、AIかな漢字変換の場合は〝NECAIK1.DRV〟、〝NECAIK2.DRV〟、単文節変換の場合は〝NECDIC.DRV〟というデバイスドライバで提供しています。日本語処理機能を利用するには、これらの日本語入力用のデバイスドライバをCONFIG.SYSファイルに組み込みます。デバイスドライバの詳細な組み込み方法については「MS–DOSユーザーズリファレンスマニュアル」を参照してください。

2.2 日本語入力用ファンクションの呼び出し方法

プログラムから日本語入力機能を利用するには、CLレジスタにファンクションの番号を入れます。またファンクション以外に必要な情報がある場合は、指定されているレジスタに必要な情報を設定して、割り込みタイプDCH（INT DCH）を実行します。

例）学習機能をなしに設定する

```
        MOV     CL,0E3H;CL＝ファンクション
        MOV     AX,0    ;学習機能なし
                        ;そのほかに、必要な情報があれば各レジスタにセットする
        INT     DCH
```

呼び出し後のレジスタの内容はリターンで定義されているレジスタ以外はすべて保存されます。定義されていないファンクションで呼び出しを行った場合は、何も実行されません。異常終了の場合はキャリーフラグがセットされた状態になります。正常終了の場合はAXレジスタの値は不定になります。

また変換を行うとき、〝読み〟の中の次の文字や記号は変換されません。

カタカナ　（読みの中のカタカナの部分）
記号類　－　「　」　、　。　・

2.3 日本語入力拡張機能ファンクション一覧

日本語入力のために次のような拡張機能が用意されています。

ファンクション	機　能
E0H	アプリケーションへの開放
E1H	アプリケーションからの使用禁止
E2H	キーボードからの日本語入力の禁止／許可
E3H	学習機能の有無を設定
E4H	ローマ字列をカナ文字列に変換
E5H	1バイト JIS 文字列を全角文字列に変換
E6H	1バイト JIS 文字列を2バイト半角文字列（シフト JIS）に変換
E7H	辞書のオープン
E8H	辞書のクローズ
E9H	語句の登録
EAH	語句の削除
EBH	語句の学習
ECH	語句の変換（単文節変換：最初の候補）
EDH	語句の変換（単文節変換：次候補）
EEH	語句の変換（単文節変換：前候補）
EFH	日本語入力モードに入る
F0H	日本語入力モードから抜ける
F1H	日本語入力モードの設定
F2H	日本語入力モードの取得
F3H	2バイト JIS をシフト JIS に変換
F4H	シフト JIS を2バイト JIS に変換
F7H	AI かな漢字変換ドライバの有無の取得
F8H	辞書の先読みと逐次変換
F9H	連文節変換（最初の候補）
FAH	連文節変換（次候補）
FBH	連文節変換（前候補）
FCH	学習（連文節）
FDH	先読み機能の有無の設定

これ以降では、各ファンクションごとにその使用方法を解説します。
なお、各機能のタイトルについているマークは次のような意味です。

（逐）……AI かな漢字変換ドライバの AI 逐次／逐次変換モードで利用できる機能
（連）……AI かな漢字変換ドライバの AI 連文節／連文節変換モードで利用できる機能
（文）……単文節変換ドライバで利用できる機能

上記のマークのついていない機能は、AI 逐次変換、逐次変換、AI 連文節変換、連文節変換、単文節変換のいずれでも利用できる機能です。

INT DCH

ファンクション E0H　アプリケーションへの開放

コール　AX ＝0

リターン　AX ＝1　日本語入力機能が使用可
　　　　　　＝0　日本語入力機能が使用不可

解　説

日本語入力機能をアプリケーションに開放します。また、日本語入力機能がシステムに組み込まれているかどうかの情報を返します。日本語入力機能を利用するときは、最初に必ずこのコールを実行しなければなりません。

AI逐次／AI連文節、逐次／連文節の機能を使用する場合は、このコールの後に必ずファンクションF7H（AIかな漢字変換ドライバの有無の取得）をコールしてください。

INT DCH

ファンクション E1H　アプリケーションからの使用禁止

コール　なし

リターン　キャリーフラグがセットされた場合
　AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
　正常終了

解　説

日本語入力機能をアプリケーションから使えないようにします。また、これをコールすると現在オープン中の辞書はクローズされます。

アプリケーションから日本語入力機能を使用した場合には、最後に必ずこのコールを実行しなければなりません。

INT DCH

ファンクション
E2H キーボードからの日本語入力の禁止／許可

コール

AX ＝0　日本語入力の許可
　　≠0　日本語入力の禁止

リターン

キャリーフラグがセットされた場合
　AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
　正常終了

解　説

キーボードからの日本語入力を、AX レジスタの値により禁止または許可します。

ファンクション E1H（アプリケーションからの日本語入力機能の使用を禁止）をコールすると、キーボードからの日本語入力は許可状態になります。

INT DCH

ファンクション

E3H 学習機能の有無を設定

コール

AX ＝0　学習機能なし
　≠0　学習機能あり

リターン

キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合

正常終了

AX ＝0　設定前は学習機能なし
　＝1　設定前は学習機能あり

解　　説

学習機能の有無を設定します。
語句の学習を行うときには、このコールで〝学習機能あり〟にセットしなければなりません。

INT DCH

ファンクション E4H　ローマ字列をカナ文字列に変換

コール

DS：SI　＝ローマ字列へのポインタ（終端は NULL）
ES：DI　＝変換結果格納域へのポインタ
AX　　　＝変換結果格納域の大きさ（バイト数）

リターン

キャリーフラグがセットされた場合
　AX　＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
　正常終了
　AX　＝変換結果の長さ（バイト数）
　CX　＝変換されたローマ字列の長さ（バイト数）

解　　説

与えられたローマ字列（1 バイト JIS）をカナ文字列（1 バイト JIS）に変換します。変換は、変換結果が変換結果格納域に納まる範囲内で行われます。

ローマ字列内に、ローマ字規則（日本語入力ガイドの付録を参照）に反する文字列や英数字以外の文字が発見された場合には、その箇所で変換を終了します。

ローマ字列の最後に〝N〞があった場合は変換されません。〝ン〞と変換したいときには〝N'〞と格納してください。また、ローマ字列の最後には NULL（00H）をセットしておいてください。

INT DCH

ファンクション E5H　1バイトJIS文字列を全角文字列に変換

コール

DS：SI ＝1バイトJIS文字列へのポインタ（終端はNULL）
ES：DI ＝変換結果格納域へのポインタ
AX ＝変換結果格納域の大きさ（バイト数）
DX ＝0　カナを全角カタカナに変換
　＝1　カナを全角ひらがなに変換

リターン

キャリーフラグがセットされた場合
　AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
　正常終了
　AX ＝変換結果の長さ（バイト数）
　CX ＝変換された1バイトJIS文字列の長さ（バイト数）

解　説

与えられた1バイトJIS文字列を全角文字列（シフトJIS）に変換します。変換は、変換結果が変換結果格納域に納まる範囲内で行われます。

1バイトJIS文字列内に1バイトJIS以外の文字（制御コード00H〜1FHと漢字80H〜9FH、E0H〜FCH）が発見された場合には、その箇所で変換を終了します。

1バイトJIS文字列の最後にはNULL（00H）をセットしてください。

濁点、半濁点は直前の文字と合わせて処理されます（カ　゛→が、ハ　゜→ぱなど）。

変換結果の格納形式は上位バイト、下位バイトの順です（全角の〝A〞（8260H）は、82H、60Hと格納されます）。

INT DCH

ファンクション
E6H 1バイトJIS文字列を2バイト半角文字列（シフトJIS）に変換

コール

DS：SI ＝1バイトコード文字列へのポインタ（終端はNULL）
ES：DI ＝変換結果格納域へのポインタ
AX ＝変換結果格納域の大きさ（バイト数）

リターン

キャリーフラグがセットされた場合
AX ＝1 日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
正常終了
AX ＝変換結果の長さ（バイト数）
CX ＝変換された1バイトJIS文字列の長さ（バイト数）

解　　説

与えられた1バイトJIS文字列（1バイトJIS）を2バイト半角文字列（シフトJIS）に変換します。変換は、変換結果が変換結果格納域に納まる範囲内で行われます。

1バイトJIS文字列内に1バイトJIS以外の文字（制御コード00H〜1FHと、漢字80H〜9FH、E0H〜FCH）が発見された場合には、その箇所で変換を終了します。

1バイトJIS文字列の最後にはNULL（00H）をセットしてください。

空白（20H）は変換されず20Hのままです。変換結果の格納形式は上位バイト、下位バイトの順です（半角の〝A〟（8560H）は85H、60Hと格納されます）。

INT DCH

ファンクション E7H 辞書のオープン

コール

DS：SI ＝辞書ファイル名へのポインタ
ES：DI ＝12バイトの領域をさすポインタ

リターン

キャリーフラグがセットされた場合

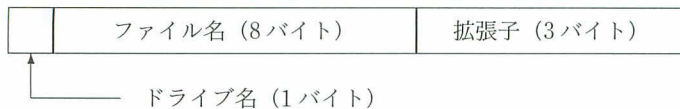
AX	＝1	日本語入力機能がアプリケーションに開放されていない
	＝3	すでに辞書がオープンしている
	＝4	指定された辞書がみつからない
	＝14	ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了
ES：DI ＝直前の辞書情報へのポインタ
（これは、オープンが成功しても失敗してもセットされる）

解　説

指定された辞書をオープンします。
DS：SIには辞書ファイル名へのポインタをセットします。
ファイル名は次の形式になっていなければなりません。

	ファイル名（8バイト）	拡張子（3バイト）

↑ ドライブ名（1バイト）

ドライブ名

0　カレントドライブ
1　ドライブA：
2　ドライブB：
：
：

ファイル名　8文字までの長さのファイル名を大文字でセットします。ファイル名はフィールドの先頭からセットし、8文字に満たない場合には残りにスペース（20H）をセットしてくだ

さい。
ファイル名の先頭が00Hの場合には、現在の辞書が選択されます。

拡張子　3文字までの長さの拡張子を大文字でセットします。
拡張子はフィールドの先頭からセットし、3文字に満たない場合には、残りにスペースをセットしてください。拡張子がない場合には、すべてスペースをセットしてください。

ES:DIには12バイトの領域を指すポインタをセットします。リターン時に、この領域には直前の辞書の情報が入ります。返される辞書の情報の形式は、DS:SIにセットしたファイル名の形式と同じです。

オープンに成功すると、指定された辞書が現在の辞書になります。失敗した場合には現在の辞書は変わりません。

INT DCH

ファンクション
E8H　辞書のクローズ

コール　なし

リターン　キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない
＝2　辞書がオープンされていない
＝14　ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了

解　説

現在オープン中の辞書をクローズします。

INT DCH

ファンクション
E9H 語句の登録

コール

DS：SI ＝読みへのポインタ（終端は NULL）
ES：DI ＝語句へのポインタ（終端は NULL）

リターン

キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない
＝2　辞書がオープンされていない
＝5　読みまたは語句が長すぎる
＝6　読みまたは語句に不正な文字が含まれている
＝9　読みの登録されるページがない
＝10　登録するための領域がない
＝14　ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合

正常終了

解　　説

オープン中の辞書に指定された読みで、指定された語句を登録します。

DS：SI には読みへのポインタをセットします。読みは 1 バイト JIS コードで、16 文字以内でなければなりません。また、読みとして〝!〞、〝*〞、〝　〞（空白）、制御コード（00H〜1FH）は使用できません。読みの最後には NULL（00H）をセットしてください。部首選択部へ登録する場合は、読みの先頭に〝ﾟ〞(DFH) を付けてください。

ES：DI には登録する語句へのポインタをセットします。登録する語句はシフト JIS で表された漢字で 16 文字以内でなければなりません。格納形式は上位バイト、下位バイトの順です（全角の〝A〞(8260H) は、82H、60H とセットします)。語句の最後には NULL（00H）をセットしてください。

このファンクションをコールした後は次候補（ファンクション EDH、FAH)、前候補（ファンクション EEH、FBH）は使用できません。また、この語句の登録で品詞情報の登録は行えません。登録語句はすべてユーザー登録の語句になります。

INT DCH

ファンクション EAH　語句の削除

コール

DS：SI ＝読みへのポインタ（終端は NULL）
ES：DI ＝語句へのポインタ（終端は NULL）

リターン

キャリーフラグがセットされた場合

AX	＝1	日本語入力機能がアプリケーションに開放されていない
	＝2	辞書がオープンされていない
	＝5	読みまたは語句が長すぎる
	＝6	読みまたは語句に不正な文字が含まれている
	＝7	読みまたは語句がみつからない
	＝11	削除不可能（システム登録の単語）
	＝14	ディスク I/O エラーが発生

キャリーフラグがセットされない場合
正常終了

解　　説

オープン中の辞書から指定された読みの、指定された語句を削除します。

DS：SI には読みへのポインタをセットします。読みは１バイト JIS コードで 16 文字以内でなければなりません。また、読みとして〝!″、〝*″、〝　″（空白）、制御コード（00H～1FH）は使用できません。読みの最後には NULL（00H）をセットしてください。部首選択部から削除する場合は、読みの先頭に〝°″(DFH）を付けてください。

ES：DI には削除する語句へのポインタをセットします。削除する語句はシフト JIS で表された漢字で 16 文字以内でなければなりません。格納形式は上位バイト、下位バイトの順です（全角の〝A″（8260H）は、82H、60H とセットします）。語句の最後には NULL（00H）をセットしてください。

このファンクションをコールした後は次候補（ファンクション EDH、FAH）、前候補（ファンクション EEH、FBH）は使用できません。また、この語句の削除ではシステム登録の単語の削除はできません。削除語句はユーザー登録の単語に限られます。

INT DCH

ファンクション

EBH 語句の学習 (文)

コール

DS：SI ＝読みへのポインタ（終端は NULL）
DS：BX＝読みへのポインタ（終端は NULL）
ES：DI ＝学習する語句へのポインタ（終端は NULL）

リターン

キャリーフラグがセットされた場合

AX ＝1	日本語入力機能がアプリケーションに開放されていない
＝2	辞書がオープンされていない
＝5	読みまたは語句が長すぎる
＝6	読みまたは語句に不正な文字が含まれている
＝7	読みまたは語句がみつからない
＝12	学習機能が未設定
＝14	ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了

解説

オープン中の辞書内にある指定された語句を指定された読みで、候補の先頭にします（学習機能）。

DS：SI には読みへのポインタをセットします。読みは 1 バイト JIS コードで 16 文字以内でなければなりません。また、読みとして〝!″、〝*″、〝 ″（空白）、制御コード（00H～1FH）は使用できません。読みの最後には NULL（00H）をセットしてください。部首選択部の語句の学習をする場合は読みの先頭に〝°″(DFH) を付けてください。

DS：BX には読みへのポインタをセットします。この読みはシフト JIS コードで 16 文字以内でなければなりません。この読みは DS：SI の指す 1 バイト JIS の読みをシフト JIS に変換したものです。格納形式は上位バイト、下位バイトの順です（全角の〝A″（8260H）は、82H、60H とセットします)。1 バイト JIS の読みと同様に〝!″、〝*″、〝 ″（空白）は使用できません。読みの最後には NULL（00H）をセットしてください。部首選択部の語句の学習をする場合は、読みの先頭に〝°″(814BH) を付けてください。

ES：DI には学習する語句へのポインタをセットします。学習する語句はシフト JIS で表された漢字で 16 文字以内でなければなりません。格納形式は、上位バイト、下位バイトの順です。語句の最後には NULL（00H）をセットしてください。

このファンクションをコールした後は次候補（ファンクション EDH、FAH)、前候補（ファンクション EEH、FBH）は使用できません。必ず最初の候補（ファンクション ECH）を呼び出してください。

INT DCH

ファンクション ECH　語句の変換（単文節変換：最初の候補）（文）

コール

DS：SI ＝読みへのポインタ（終端は NULL）
DS：BX＝読みへのポインタ（終端は NULL）
ES：DI ＝変換結果格納域へのポインタ

リターン

キャリーフラグがセットされた場合
AX ＝1　日本語入力機能がアプリケーションに開放されていない
＝2　辞書がオープンされていない
＝5　読みが長すぎる
＝6　読みに不正な文字が含まれている
＝8　候補が見つからない
＝14　ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了
AX ＝変換結果の長さ（バイト数）

解　説

指定された読みを変換して最初の候補を返します。

DS：SI には読みへのポインタをセットします。読みは 1 バイト JIS コードで 16 文字以内でなければなりません。また、読みとして〝!″、〝*″、〝　″（空白）、制御コード（00H～1FH）は使用できません。読みの最後には NULL（00H）をセットしてください。部首選択部の変換を行う場合は読みの先頭に〝°″(DFH）を付けてください。

DS：BX には読みへのポインタをセットします。この読みはシフト JIS コードで 16 文字以内でなければなりません。この読みは DS：SI の指す 1 バイト JIS の読みをシフト JIS に変換したものです。格納形式は上位バイト、下位バイトの順です（全角の〝A″（8260H）は、82H、60H とセットします)。1 バイト JIS の読みと同様に〝!″、〝*″、〝　″（空白）は使用できません。また、カタカナは変換の対象からはずされます。読みの最後には NULL（00H）をセットしてください。部首選択部の変換を行う場合は、読みの先頭に〝°″(814BH）を付けてください。

ES：DI には変換結果の格納域へのポインタをセットします。変換結果の格納形式は、上位バイト、下位バイトの順です。変換結果格納域の大きさは最大 62 バイト必要です。

INT DCH

ファンクション EDH 語句の変換（単文節変換：次候補） （文）

コール

DS：SI ＝ファンクション ECH で指定した値
DS：BX＝ファンクション ECH で指定した値
ES：DI ＝変換結果格納域へのポインタ

リターン

キャリーフラグがセットされた場合

AX ＝1	日本語入力機能がアプリケーションに開放されていない
＝2	辞書がオープンされていない
＝5	読みが長すぎる
＝6	読みに不正な文字が含まれている
＝8	候補が見つからない
＝14	ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了
AX ＝変換結果の長さ（バイト数）

解　説

指定された読みを変換し、直前に呼び出したファンクション ECH（最初の候補）、EDH（次候補）、EEH（前候補）で返された候補の次の候補を返します。

DS：SI と DS：BX にはファンクション ECH（最初の候補）で指定した値をセットします。

ES：DI には変換結果格納域へのポインタをセットします。

変換結果格納域の大きさは最大 62 バイト必要です。

INT DCH

ファンクション EEH　語句の変換（単文節変換：前候補）　（文）

コール

DS：SI ＝ファンクション ECH で指定した値
DS：BX＝ファンクション ECH で指定した値
ES：DI ＝変換結果格納域へのポインタ

リターン

キャリーフラグがセットされた場合
AX ＝１　日本語入力機能がアプリケーションに開放されていない
＝２　辞書がオープンされていない
＝５　読みが長すぎる
＝６　読みに不正な文字が含まれている
＝８　候補が見つからない
＝14　ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了
AX ＝変換結果の長さ（バイト数）

解　　説

直前に呼び出したファンクション EDH（次候補）、EEH（前候補）で返された候補の前の候補を返します。
DS：SI と DS：BX にはファンクション ECH（最初の候補）で指定した値をセットします。
ES：DI には変換結果格納域へのポインタをセットします。
変換結果格納域の大きさは最大 62 バイト必要です。

INT DCH

日本語入力モードに入る

コール　なし

リターン　キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない
＝13　キーボードからの日本語入力が禁止されている

キャリーフラグがセットされない場合

正常終了

解　　説

日本語入力モードに入ります。
これは、日本語入力モードに入るために CTRL ＋ XFER を押したときと同じ機能です。
すでに日本語入力モードに入っているときは何も実行しません。

INT DCH

ファンクション
F0H 日本語入力モードから抜ける

コール　なし

リターン　キャリーフラグがセットされた場合
AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
正常終了

解　説

日本語入力モードから抜けます。
これは、日本語入力モードから抜けるために CTRL ＋ XFER を押したときと同じ機能です。
日本語入力モードでないときは何も実行しません。

INT DCH

ファンクション F1H 日本語入力モードの設定

コール

AX = | b_{15} | b_{14} | b_{13} | b_{12} | b_{11} | b_{10} | b_9 | b_8 | b_7 | b_6 | b_5 | b_4 | b_3 | b_2 | b_1 | b_0 |

b_0	:0	直接	1	間接
b_1	:0	英数	1	カナ
b_2	:0	カタカナ	1	ひらがな
b_4b_3	:00	全角(シフト JIS)		
	01	2バイト半角(シフト JIS)		
	10	1バイト JIS		
b_5	:0		1	JIS
b_6	:0		1	部首
b_7	:0	逐次	1	連文
b_8	:予約			
b_9〜b_{15}	:未使用			

リターン

キャリーフラグがセットされた場合

AX =1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合

正常終了

AX =実際にセットされたモード

解　　説

日本語入力モードに入ったときのモードの設定をします。

モードのセットはAXレジスタの各ビットで指定します。

b_2は、$b_1=1$かつ$b_4b_3=0$のときにのみ有効です。また$b_5=1$のときは他のビットはすべて無効になります。

b_7は、AIかな漢字変換ドライバでのみ使用できます。

INT DCH

ファンクション
F2H 日本語入力モードの取得

コール　なし

リターン

キャリーフラグがセットされた場合
　AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
　正常終了
　AX ＝現在のモード
　（ファンクション F1H のものに加え）
　b_8：0　　1　AI 変換

解　　　説

日本語入力モードのときの現在のモードを取得します。

モードは、リターン時の AX レジスタの各ビットで示されます。各ビットの意味はファンクション F1H（日本語入力モードのセット）を参照してください。

b_8の“AI 変換”は取得のみ可能で、b_7と組み合わせ、AI 逐次、AI 連文節のモードであることを示します。

また、b_8はファンクション E7H（辞書のオープン）実行後にのみ有効になり、辞書ファイル（NECAI.SYS）がオープンされていないと b_8は 0 に固定されます。

INT DCH

ファンクション F3H 2バイトJISをシフトJISに変換

コール　AX ＝2バイトJISコード

リターン　キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない
　　＝6　語句に不正な文字が含まれている

キャリーフラグがセットされない場合

正常終了
AX ＝シフトJISコード

解　説

与えられた2バイトJISコードをシフトJISコードに変換します。
変換するJISコードは次の範囲内でなければなりません。

- 全角　21H ≦ AH ≦ 7EH、21H ≦ AL ≦ 7EH
- 2バイト半角　AH＝00H、21H ≦ AL ≦ 7EH または AH＝00H、A1H ≦ AL ≦ DFH

また、JISコードの29XXH、2AXXH、2BXXHも2バイト半角として扱えます（XXは21H〜7EHの16進数）。

INT DCH

ファンクション F4H　シフトJISを2バイトJISに変換

コール　AX ＝シフトJISコード

リターン　キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない
＝6　語句に不正な文字が含まれている

キャリーフラグがセットされない場合

正常終了

AX ＝2バイトJISコード

解　説

与えられたシフトJISコードを2バイトJISコードに変換します。
変換するシフトJISコードは次の範囲内でなければなりません。

・81H ≦ AH ≦ 9FH、40H ≦ AL ≦ FCH（AL ≠ 7FH）　または
・E0H ≦ AH ≦ EFH、40H ≦ AL ≦ FCH（AL ≠ 7FH）

また、シフトJISコードの85XXH（2バイト半角）（XXは40H〜FCHの16進数）はJISコードの29XXH、2AXXH、2BXXH（XXは21H〜7EHの16進数）に変換されます。

INT DCH

ファンクション
F7H AIかな漢字変換ドライバの有無の取得（逐）（連）

コール　AX ＝0

リターン

キャリーフラグがセットされた場合

AX ＝1　日本語入力機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合

AX ＝0　AIかな漢字変換ドライバはインストールされていない

＝3　AIかな漢字変換ドライバが使用可能

解　　説

AIかな漢字変換ドライバのインストールの有無を返します。

このファンクションをコールする前にファンクションE0H（アプリケーションへの開放）をコールしておく必要があります。

このコールでAXに3（0003H）が返された場合のみ、AIかな漢字変換関連の機能（ファンクションF8H～FDH）が使用可能になります。

参　考

連文節変換機能を例として、プログラムから使用する場合に参考となる処理フローを次に示します。

1.　全体的な処理の流れは次のようになります。

・アプリケーションプログラムへの開放。

・AI かな漢字変換ドライバの有無取得。

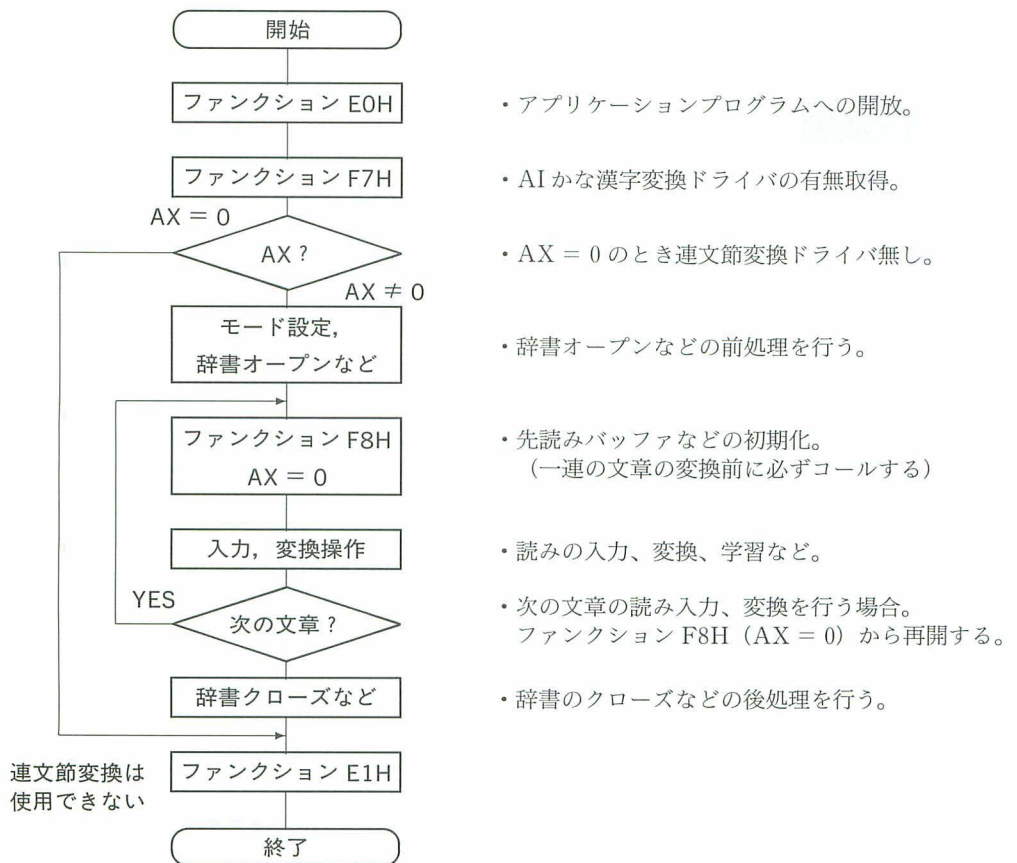
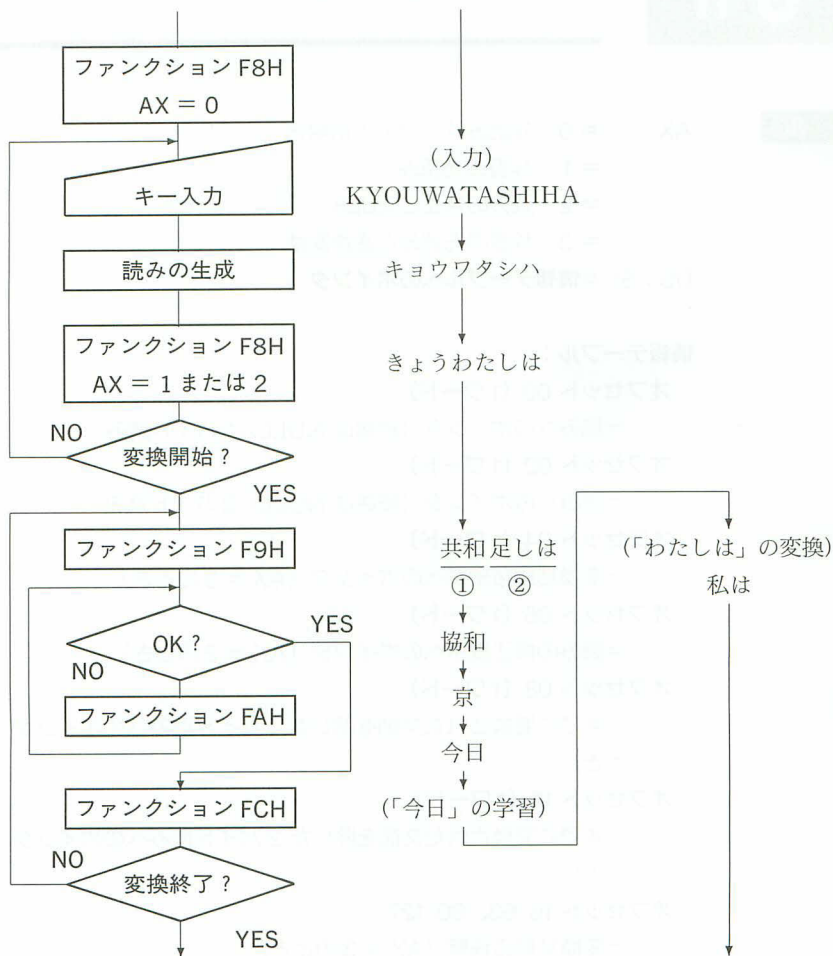
・AX＝0 のとき連文節変換ドライバ無し。

・辞書オープンなどの前処理を行う。

・先読みバッファなどの初期化。
（一連の文章の変換前に必ずコールする）

・読みの入力、変換、学習など。

・次の文章の読み入力、変換を行う場合。
ファンクション F8H（AX＝0）から再開する。

・辞書のクローズなどの後処理を行う。

2. 次に変換操作の処理の流れをフローチャートに示します。
この例では「今日私は」を変換しています。

INT DCH

ファンクション F8H　辞書の先読みと逐次変換　（逐）（連）

コール

AX　＝0　先読みバッファの初期化
　　＝1　辞書の先読み
　　＝2　読みの修正と先読み
　　＝3　辞書の先読みと逐次変換
DS：SI　＝情報テーブルへのポインタ

情報テーブル：
　オフセット00（1ワード）
　　＝読みへのポインタ（終端はNULL）1バイト読み
　オフセット02（1ワード）
　　＝読みへのポインタ（終端はNULL）2バイト読み
　オフセット04（1ワード）
　　＝変換結果格納域へのポインタ（AX＝3のとき）
　オフセット06（1ワード）
　　＝読みの修正箇所へのポインタ（AX＝2のとき）
　オフセット08（1ワード）
　　＝すでに変換された文節を除いた1バイト読みへのポインタ（AX＝3のとき）
　オフセット10（1ワード）
　　＝すでに変換された文節を除いた2バイト読みへのポインタ（AX＝3のとき）
　オフセット16–63、80–127
　　＝変換文節の情報（AX＝3のとき）

リターン

キャリーフラグがセットされた場合
　AX＝1　逐次／連文節変換機能がアプリケーションに開放されていない
　　＝2　辞書がオープンされていない
　　＝5　読みが長すぎる
　　＝6　読みに不正な文字が含まれている
　　＝8　候補が見つからない
　　＝14　ディスクの読み取り中にエラーが発生
（2、5、6、8、14は、先読みバッファの初期化（AX＝0）のコールでは返されない）

キャリーフラグがセットされない場合

AX ＝0　正常終了

または、辞書の先読み機能が設定されないので未処理（辞書の先読みと逐次変換のコール以外または、辞書の先読みと逐次変換のコールの場合は、逐次変換されなかった場合にセットされる）。

＝逐次変換された文節の長さ（バイト数）

情報テーブル：

オフセット 14（1バイト）

＝逐次変換された文節数

オフセット 16–31、80–95

＝逐次変換された文節の読み（1バイト JIS）の長さ

オフセット 32–47、96–111

＝逐次変換された文節の読み（シフト JIS）の長さ

オフセット 48–63、112–127

＝逐次変換された文節の最初の候補の長さ

解　　説

・指定された読みによる変換前の辞書の検索を行います。

辞書の先読み機能が設定されている場合、読みが1文字増えるごとにこのファンクションをコールする必要があります。

このファンクションは、ファンクション F9H（連文節変換）に先立ってコールされ、指定された読みのすべての読み方に対し辞書からデータを読み、連文節変換ドライバ内の領域に記憶しておくためのものです。

AX＝3（辞書の先読みと逐次変換）のコールは先読みを行い、記憶したデータをもとに文節作成を行って出力可能となった文節がある場合、情報テーブルに最初の候補情報を返します。情報テーブルのオフセット 14には、逐次変換された文節数の合計がセットされ、情報テーブルのオフセット 16–63、80–127には、逐次変換されたすべての各文節の長さがセットされていきます。なお、ここでセットされた情報はそのままにしておきます。

AX＝0（先読みバッファの初期化）のコールは先読みの機能の有無にかかわらず、その読みに対する最初の連文節変換または AX＝3のコールに先立って必ずコールします。情報テーブルのオフセット 00には、読みへのポインタ（1ワード）をセットします。読みは1バイト JIS で 64文字以内でなければなりません。

AX＝3と AX＝1との混在のコールはできません。また、読みには〝!″、〝*″、〝 ″（空白）、制御コード（00H～1FH）は使用できません。読みの最後には、NULL（00H）をセットしてください。濁音（〝が″など）や半濁音（〝ぱ″など）の1バイト JIS の読みは、2バイト（〝が″なら〝カ″と〝゛″）で1文字として扱ってください。

また、各情報テーブルにセットする内容は次のようになります。

・オフセット02　読みへのポインタ（1ワード）をセットします。この読みは、上記1バイトJISの読みをシフトJISに変換したものでなければなりません。また、読みの最後にはNULL（00H）をセットしてください。

・オフセット04　変換結果の格納域へのポインタをセットします。変換結果は上位バイト、下位バイトの順にセットされます。

・オフセット06　AX = 2のコールで参照されます。このコールは〝AX = 1またはAX = 3の〟コールを行っている途中、読みの一部が変更された場合にシフトJISの読みの修正箇所へのポインタとして使用します。また修正箇所へのポインタは、逐次変換された文節の読みには使用できません。情報テーブルのオフセット14の逐次変換された文節数、オフセット16–63、80–127の逐次変換された長さを確認し、逐次変換されていない読みにポインタにセットしてください。

例　1

〝わたしは〟を変換するときは、まずAX = 0のコールを行い連文節変換ドライバ内の先読みバッファを初期設定します。
次にAX = 1（先読み）のコールを続けます。

	読みへのポインタ（1バイトJIS）	読みへのポインタ（シフトJIS）
①	ワ NUL	わ NUL
②	ワタ NUL	わた NUL
③	ワタシ NUL	わたし NUL
④	ワタシハ NUL	わたしは NUL

〝わたしは〟を〝わたくしは〟に変更したとき、AX = 2のコールを使用します。

	読みへのポインタ（1バイトJIS）	読みへのポインタ（シフトJIS）
⑤	ワタクシハ NUL	わたくしは NUL

修正箇所へのポインタ（〝く〟を指す）

＊〝NUL〟は、00Hを表します。

例 2

辞書の先読みと逐次変換のコールで変換された文節が出力される場合は次のようになります。

コール例
AX ＝3
情報テーブル：

オフセット		
	00–01	ワタシハキョウノゴ NUL
	02–03	わたしはきょうのご NUL
	04–05	

リターン例
AX ＝4（4バイト）
情報テーブル：

オフセット			
	04–05	私は	（変換結果（4バイト））
	14	1（逐次変換された文節の合計）	
	16–31	04	（第1文節4バイト）
	32–47	08	（第1文節8バイト）
	48–63	04	（第1文節4バイト）

続いて、読みが増えて2文節目が出力される場合は次のようになります。

コール例
AX ＝3
情報テーブル：

オフセット		
	00–01	ワタシハキョウノゴゴウ NUL
	02–03	わたしはきょうのごごう NUL
	04–05	
	08–09	キョウノゴゴウ NUL
	10–11	きょうのごご NUL

リターン例

AX ＝６（６バイト）

情報テーブル：

オフセット　04–05　［今日の］

14　2（逐次変換された文節数の合計）

オフセット			説明
16–31	04	04	（第１文節４バイト、第２文節４バイト）
32–47	08	08	（第１文節８バイト、第２文節８バイト）
48–63	04	06	（第１文節４バイト、第２文節６バイト）

情報テーブル

オフセット

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15

オフセット	内容
00–01	読み（1バイトJIS）へのポインタ
02–03	読み（シフトJIS）へのポインタ
04–05	変換結果格納域へのポインタ
06–07	読み（シフトJIS）の修正箇所へのポインタ
08–09	すでに変換された文節を除いた1バイト読みへのポインタ
10–11	すでに変換された文節を除いた2バイト読みへのポインタ
12–13	システム予約
14	変換された文節数
15	システム予約
16–31	変換された各文節の読み（1バイトJIS）のバイト数（1～16文節）
32–47	変換された各文節の読み（シフトJIS）のバイト数（〃）
48–63	変換された各文節の漢字のバイト数（〃）
64–79	複雑文節学習時の各文節のステータス（〃）
80–95	変換された各文節の読み（1バイトJIS）のバイト数（17～32文節）
96–111	変換された各文節の読み（シフトJIS）のバイト数（〃）

112–127	変換された各文節の漢字のバイト数	（ 〃 ）
128–143	複数文節学習時の各文節のステータス	（ 〃 ）

INT DCH

ファンクション F9H　連文節変換（最初の候補）　（逐）（連）

コール

AX　　=文節番号（第１文節を０とする）
（連文節変換での最初のコールは必ず０であること。また、逐次変換ですでに変換された文節がある状態での最初のコールのときは、すでに変換されている文節数をセットする。）
DS：SI =情報テーブルへのポインタ

情報テーブル：
オフセット 00（1ワード）
=読みへのポインタ（終端は NULL）
オフセット 02（1ワード）
=読みへのポインタ（終端は NULL）
オフセット 04（1ワード）
=変換結果格納域へのポインタ
オフセット 16–63、80–127
=各文節の情報（AX が０以外のとき）

リターン

キャリーフラグがセットされた場合
AX =1　逐次／連文節変換機能がアプリケーションに開放されていない
=2　辞書がオープンされていない
=5　読みが長すぎる
=6　読みに不正な文字が含まれている
=8　候補が見つからない。または文節数が16（または32）を超えたので変換不可能
=14　ディスクの読み取り中にエラーが発生

キャリーフラグがセットされない場合
正常終了
AX =変換された文章の長さ（バイト数）

情報テーブル：
オフセット 14（バイト数）
=文節数
オフセット 16–31、80–95
=各文節の読み（1バイト JIS）の長さ
オフセット 32–47、96–111

=各文節の読み（シフト JIS）の長さ
オフセット 48–63、112–127
=各文節の最初の候補の長さ

解　　説

指定された読みに対して連文節変換を行い、最初の候補を返します。連文節変換の最初のコールでは必ず AX に 0 をセットしてコールしてください。

逐次変換ですでに変換された文節がある場合、最初のコールではすでに変換されている文節数を AX にセットしてください。

学習の後などに後続の文章を再変換する場合は、再変換する文章の先頭の文節番号をセットします。このとき DS：SI で指される情報テーブルのオフセット 16–63、80–127 には、以前のコール（ファンクション F9H～FBH）のリターン情報（各文節の読みの長さなど）をそのままセットしてください。

また、各情報テーブルにセットする内容は次のようになります。

- **オフセット 00**　読みへのポインタをセットします。読みは 1 バイトJISコードで 64 文字以内でなければなりません。また読みには〝!〞、〝*〞、〝 〞（空白）、制御コード（00H～1FH）は使用できません。読みの最後には、NULL（00H）をセットしてください。

- **オフセット 02**　シフト JIS コードで 64 文字以内の読みへのポインタをセットします。この読みは上記の 1 バイト JIS の読みをシフト JIS に変換したものでなければなりません。格納形式は上位バイト、下位バイトの順です（全角の〝A〞（8260H）は 82H、60H とセットします）。読みには 1 バイト JIS と同様に〝!〞、〝*〞、〝 〞（空白）は使用できません。また、カタカナは変換の対象からはずされています。読みの最後には必ず NULL（00H）をセットしてください。

- **オフセット 04**　変換結果の格納域へのポイントをセットします。変換結果は上位バイト、下位バイトの順にセットされます。領域の大きさは 132 バイト確保してください。

リターン時、情報テーブルのオフセット 16 以降には変換された文節数、各文節の 1 バイト JIS の読みの長さ、シフト JIS の読みの長さ、変換された漢字の長さがそれぞれ返されます。また、このコールで指定した読みは、1 バイト JIS、シフト JIS ともにその読み全体の変換が終了するまで保持しておいてください。

例　最初の連文節変換は次のように行います。

コール例

AX　＝0

情報テーブル：

オフセット		
00–01	キョウワタシハ NUL	
02–03	きょうわたしは NUL	
04–05		

リターン例

AX　＝ 10（10 バイト）

情報テーブル：

オフセット			
04–05	共和　足しは		（変換結果（10 バイト））
14	2（2 文節）		
16–31	04	03	（第 1 文節 4 バイト、第 2 文節 3 バイト）
32–47	08	06	（第 1 文節 8 バイト、第 2 文節 6 バイト）
48–63	04	06	（第 1 文節 4 バイト、第 2 文節 6 バイト）

INT DCH

ファンクション
FAH　連文節変換（次候補）　（逐）（連）

コール

AX　=文節番号（先頭文節のとき 0）
DS：SI　=情報テーブルへのポインタ

情報テーブル：
オフセット 00（1 ワード）
=ファンクション F9H で指定した値
オフセット 02（1 ワード）
=ファンクション F9H で指定した値
オフセット 04（1 ワード）
=変換結果格納域へのポインタ
オフセット 16–63、80–127
=各文節の情報

リターン

キャリーフラグがセットされた場合
AX　=1　逐次／連文節変換機能がアプリケーションに開放されていない
=2　辞書がオープンされていない
=6　読みに不正な文字が含まれている
=8　候補が見つからない
=14　ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
正常終了
AX　=次候補の長さ（バイト数）

情報テーブル：
オフセット 14（1 バイト）
=文節数
オフセット 16–31、80–95（1 バイト× 32）
=各文節の読み（1 バイト JIS）の長さ
オフセット 32–47、96–111（1 バイト× 32）
=各文節の読み（シフト JIS）の長さ
オフセット 48–63、112–127（1 バイト× 32）
=各文節の候補の長さ

解　　説

指定された番号の文節に対する次候補を返します。

AX には次候補を獲得したい文節の文節番号（ファンクション F9H、FAH、FBH）のリターン情報をもとにして）をセットします。

また、各情報テーブルにセットする内容は次のようになります。

・オフセット 00–01、02–03
　ファンクション F9H で指定した値（読みへのポインタ）をセットします。

・オフセット 04–05
　変換結果の格納域へのポインタをセットします。結果の格納方式はファンクション F9H と同じです。

・オフセット 16–63、80–127
　各文節の情報をセットします。ここには直前のファンクション F9H、FAH、FBH で得たリターン情報を変更せずにおいてください。

該当文節の読みの長さがこのコールによって変わった場合は、情報テーブル内の該当文節の読みの長さは書き換えられ、直後の文節の読みの長さを加減して余り（不足）を調整します。

例　先頭文節の〝きょうわ〟に対する次候補を獲得する場合は次のようになります。

コール例　〝共和（きょうわ）〟→〝今日（きょう）〟
AX ＝0（先頭文節）

オフセット		
00–01	キョウワタシハ NUL	
02–03	きょうわたしは NUL	
04–05	共和　足しは	（以前の結果を入れる必要はありません）

オフセット	第1文節	第2文節	
16–31	04	03	（第1文節4バイト、第2文節3バイト）
32–47	08	06	（第1文節8バイト、第2文節6バイト）
48–63	04	06	（第1文節4バイト、第2文節6バイト）

この例では、最初の候補は〝共和〟で、この文節に対して何度か次候補のコールを続けたときに得られた候補と、そのときの情報テーブル内の各フィールドの値は、次のリターン例のようになります。

リターン例

AX ＝10（10バイト）

オフセット		
04–05	今日	
16–31	03 04	（第1文節3バイト、第2文節4バイト）
32–47	06 08	（第1文節6バイト、第2文節8バイト）
48–63	04 06	（第1文節4バイト、第2文節6バイト）

このリターン例では、該当文節の読みの長さが短くなっています。そのため、情報テーブルのオフセット16の内容が書き換えられ、オフセット17の内容に余り（1）が加えられています。オフセット32とオフセット33の関係も同じです。

このあと、必要ならば第1文節を学習（ファンクションFCH）し、AX＝1（第2文節目より）として連文節変換（ファンクションF9H）をコールし、2文節目以降を再変換してください。

INT DCH

ファンクション FBH 連文節変換（前候補）　（逐）（連）

コール

AX　　=文節番号（先頭文節のとき0）
DS：SI =情報テーブルへのポインタ

情報テーブル：
　オフセット 00（1ワード）
　　=ファンクション F9H で指定した値
　オフセット 02（1ワード）
　　=ファンクション F9H で指定した値
　オフセット 04（1ワード）
　　=変換結果格納域へのポインタ
　オフセット 16–63、80–127
　　=各文節の情報

リターン

キャリーフラグがセットされた場合
　AX =1　逐次／連文節変換機能がアプリケーションに開放されていない
　　=2　辞書がオープンされていない
　　=6　読みに不正な文字が含まれている
　　=8　候補が見つからない
　　=14　ディスクの読み込み中にエラーが発生

キャリーフラグがセットされない場合
　正常終了
　AX =前候補の長さ（バイト数）

情報テーブル：
　オフセット 14（1バイト）
　　=文節数
　オフセット 16–31、80–95（1バイト× 32）
　　=各文節の読み（1バイト JIS）の長さ
　オフセット 32–47、96–111（1バイト× 32）
　　=各文節の読み（シフト JIS）の長さ
　オフセット 48–63、112–127（1バイト× 32）
　　=各文節の候補の長さ

解　説

指定された番号の文節に対する前候補を返します。

AXには前候補を獲得したい文節の文節番号（ファンクションF9H、FAH、FBH）のリターン情報をもとにして）をセットします。

また、各情報テーブルにセットする内容は次のようになります。

・オフセット00–01、02–03

ファンクションF9Hで指定した値（読みへのポインタ）をセットします。

・オフセット04–05

変換結果の格納域へのポインタをセットします。結果の格納方式はファンクションF9Hと同じです。

・オフセット16–63、80–127

各文節の情報をセットします。ここには直前のファンクションF9H、FAH、FBHで得たリターン情報を変更せずにおいてください。

該当文節の読みの長さがこのコールによって変わった場合、情報テーブル内の該当文節の読みの長さは書き換えられ、直後の文節の読みの長さが加減され不足（余り）が調整されます。

INT DCH

学習（連文節）　（逐）（連）

コール

AX　　=文節番号（先頭文節のとき0）
DX　　=学習する文節数
DS：SI =情報テーブルへのポインタ

情報テーブル：
　オフセット 00（1ワード）
　　=ファンクション F9H で指定した値
　オフセット 02（1ワード）
　　=ファンクション F9H で指定した値
　オフセット 04（1ワード）
　　=学習する語句（群）へのポインタ（終端は NULL）
　オフセット 16–63、80–127
　　=各文節の情報（ファンクション F9H などのリターン情報）

リターン

キャリーフラグがセットされた場合
　AX =1　逐次／連文節変換機能がアプリケーションに開放されていない
　　=2　辞書がオープンされていない
　　=14　ディスクの読み込み中にエラーが発生
　　=15　論理エラーが発生（情報テーブルのオフセット 64–79、128–143 を参照）

情報テーブル：
　オフセット 64–79、128–143
　　=学習に関する該当文節のステータス
　　（0　この文節は、正しく学習された）
　　　5　読みまたは語句が長すぎる
　　　6　読みまたは語句に不正な文字が含まれている
　　　7　読みまたは語句が見つからない

キャリーフラグがセットされない場合
　正常終了

解　説

指定された番号の文節から指定された文節数だけ学習します。
AX には、学習する語句の文節番号を、DX には学習する文節数をセットします。
また、各情報テーブルにセットする内容は次のようになります。

・オフセット 00–01、02–03
　ファンクション F9H で指定した文章全体の読みへのポインタをセットします。

・オフセット 04–05
　学習する語句（群）へのポインタをセットします。このときの語句は、AX（文節番号）で指した文節でなければなりません。

この機能をコールした後は、次候補（ファンクション FAH)、前候補（ファンクション FBH）は使用できません。必ず辞書の先読み（ファンクション F8H）か連文節変換（ファンクション F9H）をコールしてください。
また、連続した文節の場合は、文節の区切りも学習します。

今日歯医者へ
↓　　文節の区切り学習
今日は医者へ

興味矢崎に
↓　　文節の区切り学習
今日宮崎に

私寄り合いを
↓　　文節の区切り学習
私より愛を

例　第 2 文節の〝今日〟と第 3 文節の〝医者に〟を学習する場合は次のようになります。

コール例
AX ＝ 1（第 2 文節）
DX ＝ 2（2 つの文節）

オフセット		
	00–01	ワタシハキョウイシャニイキマス NUL
	02–03	わたしはきょういしゃにいきます NUL
	04–05	今日医者に NUL

16–31	04	03	04	04
32–47	08	06	08	08
48–63	–	04	06	08
64–79				

情報テーブルのオフセット 00–03 には、連文節変換（ファンクション F9H）で指定したもの（読みへのポインタ）をそのままセットしてあります。オフセット 04–05 には学習する語句がセットされています。

連文節変換ドライバは、オフセット 00–03 で指される読みとオフセット 16–63 の情報から語句に対する読みを確認して学習を行います。

INT DCH

先読み機能の有無の設定　（連）

コール

AX ＝0　先読み機能なし
　　≠0　先読み機能あり

リターン

キャリーフラグがセットされた場合
AX ＝1　連文節変換機能がアプリケーションに開放されていない

キャリーフラグがセットされない場合
正常終了
AX ＝0　設定前は先読み機能なし
　　＝1　設定前は先読み機能あり

解　　説

連文節変換に先立つ辞書の先読み機能の有無を設定します。

辞書の先読み（ファンクション F8H）を使用するときには、AX ≠ 0 としてこの機能をコールします。先読みを行いたくない場合は AX = 0 としてこの機能をコールしてください。初期値は〝先読み機能あり〟です。

ただし、AI 逐次／逐次変換のモードでは、このファンクションは無効です（常に〝先読みあり〟の状態になっています）。

第 3 章

マウスインターフェイス

3.1 イントロダクション

マウスは画面上のカーソルの動きを容易にコントロールできるポインティングデバイスです。マウスを机上などで自由に動かすことによって、カーソルを希望する位置へ動かすことができます。

マウスには2つのボタンがあり、押したり離したりすることができます。これによって、ソフトウェア（アプリケーションなど）とのやりとりが可能になります。たとえば、カーソルを画面上のコマンドや記号の表示されている場所へ移動してボタンを押すと、ソフトウェアがそれを調べてカーソルの位置する場所のコマンドを実行します。このように、マウスはキーボードに代わる入力装置として利用することができます。

MS–DOSでは、マウスを制御するソフトウェアを〝MOUSE.SYS〟というデバイスドライバで提供しています。マウス機能を利用するには、〝MOUSE.SYS〟をCONFIG.SYSファイルに組み込みます。デバイスドライバの詳細な組み込み方法については「MS–DOSユーザーズリファレンスマニュアル」を参照してください。

3.2 マウス用デバイスドライバのための予備知識

ここでは、マウス用デバイスドライバを使うために必要となる情報ついて説明します。

■ 接続ディスプレイの種類と解像度

マウス用デバイスドライバは、次の解像度の表示モードとディスプレイで使用できます。

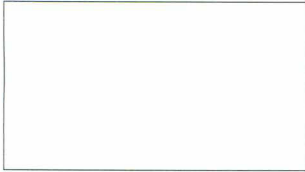
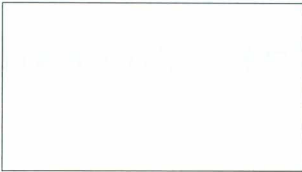
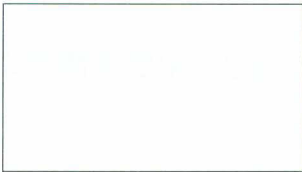
	横　　縦
ノーマルモード専用高解像度ディスプレイ	640 × 400 ドット
ノーマルモードカラーグラフィックディスプレイ	640 × 200 ドット
ハイレゾリューションモード	1120 × 750 ドット

■ 割り込みベクタ

ノーマルモードでは、割り込みベクタ番号を11H、12H、14H、15H（既定値）の中から1つ設定することができます（設定方法については「MS–DOSユーザー

ズリファレンスマニュアル」を参照してください)。

指定を省略した場合、または指定した値が上記以外のときは、既定値の 15H になります。ただしハイレゾリューションモードでは、割り込みベクタ番号は 0EH に固定されており、変更することはできません。

■ 割り込み周期

マウス用デバイスドライバでは、割り込みの周期は 4 種類をすべてサポートしています。ポート番号 BFDBH に書き込むことにより、割り込み時間を設定することができます。

割り込み周期は、短いほどマウスの動きを正確にとらえますが、デバイスドライバのオーバーヘッドが大きくなるので注意が必要です。

■ 画面の座標系

マウス用デバイスドライバには、画面の表示モードと接続されているディスプレイの組み合わせによって、3 種類の画面モードがあります。

どの画面モードでも、スクリーン上の各点（ドット）は、1 ドットを単位とした水平方向（横）と垂直方向（縦）の座標で表現します。たとえば、水平座標が 5 ドット目で、垂直座標が 10 ドット目の点の座標は（5、10）と表します。画面モードごとに使用する座標の範囲（4 隅の点の座標）は次のようになります。

■ 表示画面

マウス用デバイスドライバでは、画面モードによって表示するグラフィック用画面（VRAM）が異なります。

ノーマルモードの高解像度ディスプレイでは、次に示すグラフィック用 VRAM（GVRAM0、GVRAM1、GVRAM2、GVRAM3）の、それぞれの開始アドレスから 32K バイトずつを、GDC 合成によって画面に表示します。

グラフィック用 VRAM の開始アドレス
GVRAM0　A8000H
GVRAM1　B0000H
GVRAM2　B8000H
GVRAM3　E0000H　（オプション）

ノーマルモードのカラーグラフィックディスプレイでは、上記のグラフィック用 VRAM の、それぞれの開始アドレスから 16K バイトずつを GDC 合成によって画面に表示します。

ハイレゾリューションモードでは、C000H から DEFF0H までの 128K バイトの画面を 4 バンク持ち、GDC 合成によって画面に表示します。

■ マウスカーソル

表示できるマウスカーソルの大きさは、画面モードによって次のように異なります。この四角形の範囲内で自由な形状に、マウスカーソルをデザインすることができます。

	横　縦
ノーマルモードの高解像度ディスプレイ	16 × 32 ドット
ノーマルモードのカラーグラフィックディスプレイ	16 × 16 ドット
ハイレゾリューションモード	16 × 32 ドット

マウスカーソルには〝中心点〟という点があり、マウスカーソルの形状とともに設定します。これは、マウスカーソルの位置（座標）を設定して取得するための基準となる点で、上記の範囲の四角形の左上隅を（0,0）とする座標で表します。たとえば、左上を向いた矢印では矢印の先端の点を中心点とし、十字形では縦横の線の交点を中心点に設定します。

■ ミッキー

机上でのマウスの移動距離は、〝ミッキー〟という単位で表します。1 ミッキーは約 100 分の 1 インチ（約 0.25mm）になります。マウス用デバイスドライバでは、マウスカーソルを画面上で 8 ドット移動させるために必要な、机上でのマウスの移動距離（ミッキー単位）を設定することで、マウスの感度を変えることができます。

3.3　マウス用ファンクションの呼び出し方法

マウス用のファンクションを利用するには、AX レジスタに使用したいファンクションの機能コードと、その他の必要な情報を各レジスタに設定して、割り込みタイプ 33H（INT 33H）を実行します。

3.4　マウス用ファンクション一覧

マウス用デバイスドライバには次のような機能が用意されています。

ファンクション	機　　能
00H	環境のチェック
01H	カーソル表示
02H	カーソル消去
03H	カーソル位置の取得
04H	カーソル位置の設定
05H	左ボタンの押下情報の取得
06H	左ボタンの開放情報の取得
07H	右ボタンの押下情報の取得
08H	右ボタンの開放情報の取得
09H	カーソルの形の設定
0BH	マウスの移動距離の取得
0CH	ユーザー定義サブルーチンのコール条件の設定
0FH	ミッキー／ドット比の設定
10H	水平方向のカーソル移動範囲の設定
11H	垂直方向のカーソル移動範囲の設定
12H	カーソルの表示画面の設定
13H	グラフィック用 VRAM の設定と実装状況の取得

これ以降では、各ファンクションごとにその使用方法を解説します。

INT 33H

ファンクション **00H**

環境のチェック

コール AX ＝ 00H

リターン AX ＝環境状態
0 マウスが使用不可能
–1 マウスが使用可能

解　説

マウスを使用できる環境かどうかを調べます。

マウスが使用できる環境とは、本体にマウスインターフェイスが内蔵されているか、マウスインターフェイスボードが接続されていて、マウス用デバイスドライバがメモリ中に存在している状態のことです。アプリケーションは、他のファンクションをコールする前に、このファンクションによって環境を調べる必要があります。

また、このファンクションはカーソルの表示、カーソルの形状、カーソルの中心点、ミッキー／ドット比、ユーザー定義サブルーチンのコール条件を初期値に設定します。

INT 33H

ファンクション 01H　カーソル表示

コール　AX ＝ 01H

リターン　なし

解　説

マウスカーソルを画面に表示します。

一度このファンクションをコールすると、ファンクション 02H（カーソル消去）がコールされるまで、マウスカーソルはマウスの動きに従って画面上を動きます。

ただし、グラフィック画面が非表示になっている場合は、本ファンクションを実行してもマウスカーソルは表示されません。この場合はマウスドライバを組み込み直すか、グラフィックスドライバなどを用いてグラフィック画面を表示状態にしてください。

INT 33H

ファンクション 02H カーソル消去

コール　AX ＝02H

リターン　なし

解説

画面上に表示されているマウスカーソルを消去します。

一度このファンクションをコールすると、ファンクション 01H（カーソル表示）がコールされるまで、マウスカーソルは画面に表示されません。ただし、カーソルは表示されていない間でも、マウスを動かすことによって画面上を移動しています。

INT 33H

ファンクション 03H　カーソル位置の取得

コール

AX ＝ 03H

リターン

AX ＝左ボタンの状態
　0　押されていない
　–1　押されている
BX ＝右ボタンの状態
　0　押されていない
　–1　押されている
CX ＝マウスカーソルの位置の水平座標
　0〜639　ノーマルモードの場合
　0〜1119　ハイレゾリューションモードの場合
DX ＝マウスカーソルの位置の垂直座標
　0〜199　ノーマルモード（カラーグラフィックディスプレイの場合）
　0〜399　ノーマルモード（専用高解像度ディスプレイの場合）
　0〜749　ハイレゾリューションモードの場合

解　説

現在のマウスカーソルの位置（座標）を取得します。

カーソルの位置は、水平座標（CX）と垂直座標（DX）で取得され、値はそのときの表示モードでマウスカーソルが移動できる範囲内です。

またこのファンクションでは、そのときのマウスの左右のボタンの状態（押されているか、押されていないか）も取得することができます。

INT 33H

ファンクション
04H カーソル位置の設定

コール

AX ＝04H
CX ＝カーソルの新しい位置の水平座標
　0〜639　ノーマルモードの場合
　0〜1119　ハイレゾリューションモードの場合
DX ＝カーソルの新しい位置の垂直座標
　0〜199　ノーマルモード（カラーグラフィックディスプレイの場合）
　0〜339　ノーマルモード（専用高解像度ディスプレイの場合）
　0〜749　ハイレゾリューションモードの場合

リターン

なし

解　　説

マウスカーソルの位置を指定した位置に設定（移動）します。

アプリケーションは希望する位置の水平座標と垂直座標を指定して、このファンクションをコールするとその位置にマウスカーソルが移動します。指定した位置がマウスカーソルの移動範囲外の場合には、移動範囲内の端にカーソルを移動します。

INT 33H

ファンクション 05H 左ボタンの押下情報の取得

コール　AX ＝ 05H

リターン　AX ＝左ボタンの状態
　0　押されていない
　–1　押されている
BX ＝左ボタンが押された回数
CX ＝最後に左ボタンが押されたときのカーソル位置の水平座標
DX ＝最後に左ボタンが押されたときのカーソル位置の垂直座標

解　説

マウスの左ボタンの押下（押されている）状態に関する各種の情報を取得します。
取得する情報は次のとおりです。

・左ボタンの現在の状態（押されているか、押されていないか）
・このファンクションが最後にコールされてから、今回コールされるまでに左ボタンが押された回数
・最後に左ボタンが押されたときの、カーソルの位置（水平座標、垂直座標）

INT 33H

ファンクション
06H 左ボタンの開放情報の取得

コール

AX ＝ 06H

リターン

AX ＝左ボタンの状態
 0 押されていない
 −1 押されている
BX ＝左ボタンが離された回数
CX ＝最後に左ボタンが離されたときのカーソル位置の水平座標
DX ＝最後に左ボタンが離されたときのカーソル位置の垂直座標

解説

マウスの左ボタンの開放（離されている）状態に関する各種の情報を取得します。
取得する情報は、次のとおりです。

- 左ボタンの現在の状態（押されているか、押されていない）
- このファンクションが最後にコールされてから、今回コールされるまでに左ボタンが離された回数
- 最後に左ボタンが離されたときの、カーソルの位置（水平座標、垂直座標）

INT 33H

ファンクション 07H 右ボタンの押下情報の取得

コール　AX ＝ 07H

リターン　AX ＝右ボタンの状態
　0　押されていない
　–1　押されている
BX ＝右ボタンが押された回数
CX ＝最後に右ボタンが押されたときのカーソル位置の水平座標
DX ＝最後に右ボタンが押されたときのカーソル位置の垂直座標

解　説

マウスの右ボタンの押下（押されている）状態に関する各種の情報を取得します。
取得する情報は次のとおりです。

・右ボタンの現在の状態（押されているか、押されていない）
・このファンクションが最後にコールされてから、今回コールされるまでに右ボタンが押された回数
・最後に右ボタンが押されたときの、カーソルの位置（水平座標、垂直座標）

INT 33H

ファンクション
08H 右ボタンの開放情報の取得

コール　AX ＝ 08H

リターン　AX ＝右ボタンの状態
　0　押されていない
　−1　押されている
BX ＝右ボタンが離された回数
CX ＝最後に右ボタンが離されたときのカーソル位置の水平座標
DX ＝最後に右ボタンが離されたときのカーソル位置の垂直座標

解説

マウスの右ボタンの開放（離されている）状態に関する各種の情報を取得します。
取得する情報は、次のとおりです。

・右ボタンの現在の状態（押されているか、押されていない）
・このファンクションが最後にコールされてから、今回コールされるまでに右ボタンが離された回数
・最後に右ボタンが離されたときの、カーソルの位置（水平座標、垂直座標）

INT 33H

ファンクション 09H カーソルの形の設定

コール

AX　　= 09H
BX　　=カーソルの中心点の水平座標（0〜15）
CX　　=カーソルの中心点の垂直座標
　　0〜15　ノーマルモード（カラーグラフィックディスプレイの場合）
　　0〜31　ノーマルモード（高解像度ディスプレイの場合）
　　　　　　またはハイレゾリューションモード
ES：DX=カーソルの形状のデータのアドレス

リターン

なし

解　説

マウスカーソルの形状と中心点を設定します。

カーソルの形状は四角形のドットの集合のうち、どのドットを表示させるかによって決まります。つまり、表示されたドットの集合がカーソルとして見えます。

ユーザーはプログラムによって、四角形の範囲内で自由にカーソルの形状をデザインすることができます。たとえば、四角形のカーソルのドットの集合のうちすべてのドットを表示するように指定すると、カーソルの形は四角形になります。

データの形式は、接続されているディスプレイによって次のように異なります。

16×16ビット　ノーマルモード（カラーグラフィックディスプレイの場合）
16×32ビット　ノーマルモード（高解像度ディスプレイの場合）
16×32ビット　ハイレゾリューションモードの場合

例

カーソルの形状を決定するデータの形式は次のようになります。

```
DB 11000000B, 00000000B    ↑
DB 11100000B, 00000000B    ノーマルモード（カラーグラフィックディス
・                          プレイの場合）は16行。ノーマルモード（高
・                          解像度ディスプレイの場合）とハイレゾリュ
・                          ーションモードは32行。
DB 00000000B, 00000000B    ↓
```

カーソルの中心点は、カーソルの左上隅の点を原点（0,0）とした座標で指定します。この点は、カーソルの位置を検出する場合などに使用される点です。

なお、指定を省略した場合はシステムの初期設定によって、形状は左上向きの矢印、中心点は（0,0）のカーソルの形状になります。

INT 33H

ファンクション

0BH マウスの移動距離の取得

コール　AX ＝ 0BH

リターン　CX ＝マウスの水平方向の移動距離
　　–32768〜32767　ノーマルモード
　　–1119〜1119　ハイレゾリューションモード
　DX ＝マウスの垂直方向の移動距離
　　–32768〜32767　ノーマルモード
　　–935〜935　ハイレゾリューションモード

解　説

ミッキー単位でのマウスの移動距離を取得します。このファンクションが最後にコールされたときのマウスの位置から、今回コールされたときのマウスの位置までの、垂直方向と水平方向の相対的な距離をアプリケーションに通知します。

水平方向では右の向きを正とし、垂直方向では下向きを正とします。また、通知される距離の単位はミッキーです。

INT 33H

ファンクション **0CH**

ユーザー定義サブルーチンのコール条件の設定

コール

AX =0CH
CX =コール条件
ビット0 カーソル位置が変化した場合
1 左ボタンが押された場合
2 左ボタンが離された場合
3 右ボタンが押された場合
4 右ボタンが離された場合
5〜15 未使用
（ビット0を最下位ビットとして各ビットが1のときにコールし、0のときにはコールしない）
ES：DX＝ユーザー定義サブルーチンのアドレス

リターン

なし

解説

ユーザー（アプリケーション）が定義したサブルーチンを、マウス用デバイスドライバがコールする条件と、そのサブルーチンのアドレスを設定します。マウス用デバイスドライバでは、次の5種類の現象のひとつが発生したときにサブルーチンをコールすることができます。

・カーソル位置が変化した場合
・左ボタンが押された場合
・左ボタンが離された場合
・右ボタンが押された場合
・右ボタンが離された場合

これらの5種類のコール条件は同時に複数設定することができ、そのうちの1つが発生するとデバイスドライバはサブルーチンをコールします。

マウス用デバイスドライバからサブルーチンがコールされる手順は次のとおりです。

1. マウスからの割り込みが発生すると、制御がマウス用デバイスドライバに移る。

2. マウス用デバイスドライバは、サブルーチンのコール条件が満たされているか調べる。 コール条

件が満たされていなければ、次の処理へ進む。

3. コール条件のひとつが満たされていれば、CALL FAR–PROC命令によってサブルーチンへ制御を移す。

・

・

（サブルーチンによる処理）

・

・

4. サブルーチンはRET FAR–PROC命令によって、マウス用デバイスドライバへ制御を戻す。

マウス用デバイスドライバからサブルーチンがコールされるとき、各レジスタには次のような情報が格納されています。

AX ＝コールの原因となった条件
　1　カーソルの位置が変化した
　2　左ボタンが押された
　3　左ボタンが離された
　4　右ボタンが押された
　5　右ボタンが離された
BL ＝左ボタンの状態
　0　押されていない
　–1　押されている
BH ＝右ボタンの状態
　0　押されていない
　–1　押されている
CX ＝カーソル位置の水平座標
DX ＝カーソル位置の垂直座標

注意　マウス用デバイスドライバは、CALL FAR–PROC命令によってサブルーチンをコールします。したがって、サブルーチンからマウス用デバイスドライバへ制御を戻すときは、RET FAR–PROC命令を使用しなければいけません。

INT 33H

ファンクション 0FH ミッキー／ドット比の設定

コール

AX ＝0FH
CX ＝水平方向のミッキー／ドット比
DX ＝垂直方向のミッキー／ドット比

リターン

なし

解　　説

マウスの移動距離と画面上のカーソルの移動距離の比を設定します。

画面上でカーソルが8ドット移動するために必要なマウスの水平方向および垂直方向の移動距離を設定します。マウスの移動距離を設定する単位はミッキーで、1ミッキーは約100分の1インチ（約0.25mm）です。

ミッキー／ドット比を小さく設定すると、マウスを少し動かしただけでカーソルは大きく移動し、大きく設定するとマウスをかなり動かしてもカーソルは少ししか移動しません。このようにミッキー／ドット比の設定によって、マウスの感度を変えることができます。

設定を省略した場合のシステムの初期値は、水平方向、垂直方向ともにミッキー／ドット比は8になります。

INT 33H

ファンクション 10H 水平方向のカーソル移動範囲の設定

コール

AX ＝10H
CX ＝水平方向の移動範囲の最小値
　0〜639　ノーマルモード
　0〜1119　ハイレゾリューションモード
DX ＝水平方向の移動範囲の最大値
　0〜639　ノーマルモード
　0〜1119　ハイレゾリューションモード

リターン

なし

解　説

画面上でカーソルの中心点が移動できる水平方向（左右）の範囲を設定します。

移動範囲は最小値（左側）と最大値（右側）で設定します。CX レジスタの値が DX レジスタの値よりも大きい場合は DX レジスタの値が最小値、CX レジスタの値が最大値となります。

設定を省略した場合のシステムの初期設定値は画面全体です。

マウスが大きく動き、カーソルの位置が移動範囲外になったときは移動範囲内の端にカーソルは表示されます。

INT 33H

ファンクション 11H 垂直方向のカーソル移動範囲の設定

コール

AX ＝11H
CX ＝**垂直方向の移動範囲の最小値**
　0〜199　ノーマルモード（カラーグラフィックディスプレイの場合）
　0〜399　ノーマルモード（高解像度ディスプレイの場合）
　0〜749　ハイレゾリューションモード
DX ＝**垂直方向の移動範囲の最大値**
　0〜199　ノーマルモード（カラーグラフィックディスプレイの場合）
　0〜399　ノーマルモード（高解像度ディスプレイの場合）
　0〜749　ハイレゾリューションモード

リターン

なし

解　説

画面上でカーソルの中心点が移動できる垂直方向（上下）の範囲を設定します。

移動範囲は最小値（上側）と最大値（下側）で設定します。CX レジスタの値が DX レジスタの値よりも大きい場合は DX レジスタの値が最小値、CX レジスタの値が最大値となります。

設定を省略した場合のシステムの初期設定は画面全体です。

マウスが大きく動き、カーソルの位置が移動範囲外になったときは移動範囲内の端にカーソルは表示されます。

INT 33H

ファンクション 12H カーソルの表示画面の設定

コール

AX ＝12H
BX ＝表示画面

通　　常	PC–H98 で 256 色使用時
0：プレーン 0 へ表示	0：プレーン 0 へ表示
1：プレーン 1 へ表示	1：プレーン 1 へ表示
2：プレーン 2 へ表示	2：プレーン 2 へ表示
3：プレーン 3 へ表示	3：プレーン 3 へ表示
	4：プレーン 4 へ表示
	5：プレーン 5 へ表示
	6：プレーン 6 へ表示
	7：プレーン 7 へ表示

リターン

なし

解　　説

カーソルの表示画面（プレーン）を設定します。

カーソルの色は表示画面のパレットで設定された色になります。

PC–H98 以外の機種でプレーン 4 以降に表示画面を指定した場合は、最大プレーン（プレーン 3 または 2）に表示されます。

PC–H98 でプレーン 4 以降に表示する場合は、ファンクション 13H（グラフィック用 VRAM の設定と実装状況の取得）で VRAM を 8 プレーンまで使用するモードにしておく必要があります。このファンクションを実行せずにプレーン 4 以降への表示を指定した場合は、前回の表示画面へカーソルを表示します。

INT 33H

ファンクション **13H**

グラフィック用 VRAM の設定と実装状況の取得

コール

AX ＝13H
BX ＝グラフィック VRAM の設定
　0　0～2 プレーンを使用する
　1　0～3 プレーンを使用する（*1）
　2　0～7 プレーンを使用する（*2）

リターン

BX ＝グラフィック VRAM の実装状況
　0　0～2 プレーンを実装
　–1　0～3 プレーンを実装（*1）
　–2　0～7 プレーンを実装（*2）

（*1）ハイレゾリューションモードでは、常にプレーン 0～3 を使用できます。
（*2）PC–H98 で 256 色オプションボード（PC–H98–E02）を実装している場合のみプレーン 4～7 を使用できます。

解　説

グラフィック用 VRAM の使用するプレーンを設定します。
グラフィック用 VRAM のプレーン 3～7 が未実装の場合は、次のように動作します。

・プレーン 3 以降未実装時：プレーン 0～2 を使用する
・プレーン 4 以降未実装で、プレーン 0～7 使用要求時：直前のグラフィック VRAM の設定状態になる

3.5　各パラメータの初期値

マウス用デバイスドライバの、種々の設定を省略した場合の設定値（初期値）は次のとおりです。

カーソル表示　　　：表示しない
カーソル位置　　　：画面の中心

・ノーマルモード（カラーグラフィックディスプレイの場合）	(319,99)
・ノーマルモード（高解像度ディスプレイの場合）	(319,199)
・ハイレゾリューションモード	(512,384)

カーソルの形状　　：左上向きの矢印
カーソルの中心点　：(0,0)
カーソルの表示画面　：ノーマルモード　　　　　　プレーン２
　　　　　　　　　　ハイレゾリューションモード　プレーン０
カーソルの移動範囲　：画面全体

	水平	垂直
・ノーマルモード（カラーグラフィックディスプレイの場合）	0～639	0～199
・ノーマルモード（高解像度ディスプレイの場合）	0～639	0～399
・ハイレゾリューションモード	0～1119	0～749

ミッキー／ドット比　：8（水平方向、垂直方向ともに）

第4章

グラフィックスドライバ

4.1 イントロダクション

MS-DOSでは、PC-9800シリーズのグラフィック機能を活用するための、基本的な描画機能を収めたグラフィックスライブラリをデバイスドライバとして提供しており、アプリケーションやユーザープログラムで利用することができます。

提供されるグラフィックスドライバは、〝GRAPH.SYS〟と〝GRAPH.LIB〟の2つのファイルで構成されています。

グラフィックス機能を利用するには、〝GRAPH.LIB〟をカレントドライブのカレント（またはルート）ディレクトリに格納して、〝GRAPH.SYS〟をCONFIG.SYSファイルに組み込みます。デバイスドライバの詳細な組み込み方法については「MS-DOSユーザーズリファレンスマニュアル」を参照してください。

4.2 グラフィックスファンクションの呼び出し方法

グラフィックスドライバの各ファンクションは次の手順で呼び出します。

1. グラフィックスドライバのエントリテーブルの先頭アドレスを取得します（エントリテーブルの詳細はグラフィックスファンクション一覧を参照してください）。
   AXレジスタ、DS：BXレジスタを次のようにセットし割り込みタイプCDH（INT CDH）を行うと、DS：BXが指す領域（DWORD）にエントリテーブルの先頭アドレス（上位ワードにセグメント、下位ワードにオフセット）が格納されます。

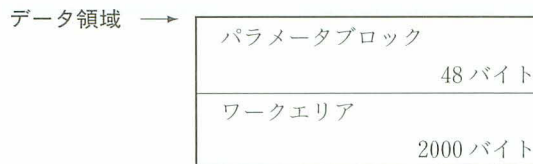
   AX　　　＝0
   DS：BX　＝任意のアドレス（セグメントとオフセット）

   ただし、グラフィックスドライバが組み込まれていない場合は、動作は保障されないで、次のデバイス名でオープン（INT21H、ファンクション3DH）して、グラフィックスドライバの組み込みの有無を確認後、INT CDHを実行してください。

デバイス名：GRAPH△△$
（△印は空白一文字を表します。）

2. 次にグラフィックスドライバで使用するデータ領域を、次のような形式で確保します。

データ領域 ──→

パラメータブロック 48バイト
ワークエリア 2000バイト

3. ファンクションを呼び出します。
2.で確保したデータ領域内のパラメータブロックに、各ファンクションで必要なパラメータを設定します。次にデータ領域の先頭アドレスをセグメント、オフセットの順にスタックに格納して、エントリテーブル内の対応するアドレスをfar callします。

なおプログラムの使用例については4.6「プログラム例」を参照してください。

4.3 グラフィックスファンクション一覧

グラフィックスデバイスドライバには次のような機能が用意されています。
また、各ファンクションのエントリテーブルの詳細もあわせて示しています。各テーブルにはそのファンクションのエントリアドレスが格納されています。

ファンクションNo		アドレス	機　能
初期化	0	0000	グラフィックの開始
	1	0004	グラフィックの終了
	2	0008	仮想VRAMの生成
環境設定	3	000C	表示モードの設定
	4	0010	描画プレーンの設定
	5	0014	表示プレーンの設定
	6	0018	パレットの設定
	7	001C	ビューポート領域の設定
	8	0020	フォアグラウンドカラーの設定
	9	0024	バックグラウンドカラーの設定
	10	0028	ボーダーカラーの設定
	11	002C	表示スイッチの設定
	12	0030	表示領域の設定
	13	0034	中断処理ルーチンの設定

ファンクションNo		アドレス	機　　能
描　　画	14	0038	画面消去
	15	003C	点の描画
	16	0040	線の描画
	17	0044	三角形の描画
	18	0048	長方形の描画
	19	004C	台形の描画
	20	0050	円の描画
	21	0054	楕円の描画
	22	0058	閉領域の塗りつぶし
	23	005C	グラフィックイメージの取得
	24	0060	グラフィックイメージの設定
	25	0064	領域転送
	26	0068	領域移動
環境取得	27	006C	バージョンの取得
	28	0070	プレーン数の取得
	29	0074	表示モードの取得
	30	0078	描画プレーンの取得
	31	007C	表示プレーンの取得
	32	0080	パレットの取得
	33	0084	ビューポート領域の取得
	34	0088	フォアグラウンドカラーの取得
	35	008C	バックグラウンドカラーの取得
	36	0090	ボーダーカラーの取得
	37	0094	表示スイッチの取得
	38	0098	指定座標のパレットの取得
	39	009C	表示領域の取得
	40	00A0	中断処理ルーチンの取得

これ以降では、各ファンクションごとにその使用方法を解説します。

INT CDH

ファンクション
No.0 グラフィックの開始

コール　スタック＝データ領域の先頭アドレス

リターン　AX ＝0　常に正常終了

解　説

グラフィック専用ハードウェア、インタフェースの初期設定をして実行環境を整えます。

グラフィックスドライバを使用する場合は、必ずこのファンクションを最初に実行してください。なお、このファンクションは画面消去は行いません。画面を消去する場合はファンクション No.14（画面消去）を必要に応じて実行してください。

具体的な初期化内容（システム VRAM および各種設定）は次のとおりです。

	ノーマルモード	ハイレゾリューションモード
表示モード（解像度 / カラーモード）	640 × 200 8色/8色	1120 × 756 16色/4096色
描画プレーン	ページ0、プレーン0〜2	ページ0、プレーン0〜3
表示プレーン	ページ0、プレーン0〜2	ページ0、プレーン0〜3
パレット	8色/8色のパレットの初期値(*)	16色/4096色のパレットの初期値(*)
ビューポート領域	(0,0) − (639,199)	(0,0) − (1119,935)
フォアグラウンドカラー	パレット番号7	
バックグラウンドカラー	パレット番号0	
ボーダーカラー	ブラック	
表示スイッチ	非表示	
表示領域	──	Y座標0〜749

(*) 詳細はファンクション No.6（パレットの設定）を参照してください。

仮想 VRAM の初期化は、ファンクション No.2（仮想 VRAM の生成）の実行時に行われます。

INT CDH

ファンクション No.1 グラフィックの終了

コール　スタック＝データ領域の先頭アドレス

リターン　AX ＝０　常に正常終了

解　　説

グラフィック専用ハードウェアをリセットします。

グラフィックスドライバの利用を終了する場合は、このファンクションを実行します。これによりグラフィックスドライバで設定したハードウェアの状態が、この後に動作するプログラムに影響しないようになります。

INT CDH

ファンクション
No.2 仮想 VRAM の生成

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	仮想 VRAM 構造体のアドレス
4H	DWORD	仮想 VRAM のアドレス
8H	WORD	X 方向のドット数
0AH	WORD	Y 方向のドット数
0CH	WORD	プレーン数

●仮想 VRAM 構造体のアドレス

仮想 VRAM 構造体は、仮想 VRAM のアドレスやプレーン数などの情報を格納しておく場所で、あらかじめ 64 バイト確保しておきます。この仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

●仮想 VRAM のアドレス

あらかじめ確保しておいた仮想 VRAM のアドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。仮想 VRAM のサイズは次の式で求められます。

仮想 VRAM のサイズ（バイト数）
＝（(X 方向ドット数＋15）/16）× 2 ×縦方向ドット数×プレーン数

このサイズは 64K を超えてもかまいません。ただし、このファンクションではメモリの管理を行いません。サイズが 64K バイトを超える場合、連続した位置に確保しておくことが必要です。

● X 方向/Y 方向のドット数

仮想 VRAM の大きさを次の範囲で指定します。

0<X 方向/Y 方向ドット数≦ 1120

●プレーン数

仮想 VRAM のプレーン数を指定します。指定したプレーン数によって、その後

この仮想 VRAM に設定できるカラーモードが次のように異なります。カラーモードについてはファンクション No.3（表示モードの設定）を参照してください。

プレーン数＼カラーモード	モノクロ	8色	16色	256色
1	○	×	×	×
3	○（*1）	○（*1）	×	×
4	○	○（*1）	○	×
8（*2）	○	○（*1）	○	○

8色　：8色/8色および8色/4096色
16色　：16色/4096色
256色　：256色/1600万色

（*1）ハイレゾリューションモードでは設定できません。
（*2）プレーン数8は、PC-H98で256色オプションボード（PC-98H-E02）を実装した装置で、専用高解像度版グラフィックスドライバを利用している場合のみ指定できます。

リターン

AX ＝0　正常終了
　 ≠0　異常終了
　指定のプレーン数が4を超えたときなどに異常終了となります。

解　説

メモリ上に仮想的なVRAMを生成して初期化を行います。
仮想VRAMと仮想VRAM構造体はあらかじめ確保しておいてください。
初期状態はノーマルモード、ハイレゾリューションモードとも次のようになります。仮想VRAMでは解像度による区別はなく、ページは0固定になります。

表示モード　：モノクロ
描画プレーン　：ページ0、プレーン0
ビューポート領域　：(0,0) －（X方向ドット数-1,Y方向ドット数-1）
フォアグラウンドカラー　：パレット番号7
バックグラウンドカラー　：パレット番号0
仮想VRAM　：任意のサイズ
仮想VRAM構造体　：64バイト

仮想VRAMに対しては、環境設定ファンクション、環境取得ファンクション、およびファンクション No.25（領域転送）が実行できます。

No.2

INT CDH

ファンクション

No.3 表示モードの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	対象 VRAM
4H	WORD	表示モード

●対象 VRAM

システム VRAM の表示モードを設定する場合は０、仮想 VRAM の表示モードを設定する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

●表示モード

	パラメータ値	表示モード	
		解像度	カラーモード
ノーマルモード	0000H	640 × 200	モノクロ
	0001H	640 × 200	8 色/8 色
	0002H	640 × 200	8 色/4096 色
	0003H	640 × 200	16 色/4096 色
	0100H	640 × 400	モノクロ
	0101H	640 × 400	8 色/8 色
	0102H	640 × 400	8 色/4096 色
	0103H	640 × 400	16 色/4096 色
	0104H	640 × 400	16 色/1600 万色（*）
	0105H	640 × 400	256 色/1600 万色（*）
ハイレゾリューションモード	0300H	1120 × 750	モノクロ
	0303H	1120 × 750	16 色/4096 色
	0304H	1120 × 750	16 色/1600 万色（*）
	0305H	1120 × 750	256 色/1600 万色（*）

モノクロ　　　：ブラック、ホワイトの２色
８色/８色　　：８色を同時に使用可能
８色/4096 色　：4096 色のうち８色を同時に使用可能

16色/4096色	：4096色のうち16色を同時に使用可能
16色/1600万色	：1600万色のうち16色を同時に使用可能
256色/1600万色	：1600万色のうち256色を同時に使用可能

（*）PC–H98で専用高解像度版グラフィックスドライバを利用している場合のみ指定できます。また、256色/1600万色のカラーモードは256色オプションボード（PC–H98–E02）を実装した装置でのみ指定できます。

リターン

AX ＝0 **正常終了**
≠0 **異常終了** （エラーコード一覧参照）
ハードウェア的に設定できないモードを指定した場合は異常終了になります。

解　説

システムVRAMまたは仮想VRAMの表示モードの設定を行います。

このファンクションを実行すると、描画プレーン、表示プレーン、ビューポート、フォアグラウンドカラー、バックグラウンドカラー、ボーダーカラー、表示領域が初期化されます。これらの初期状態については、対応する環境設定ファンクションを参照してください。

パレットの状態は前回のそのモードでの値になります。また、仮想VRAMの表示モードを設定する場合は解像度は無視されます。

ノーマルモードでモノクロモードを利用する場合、グラフ表示色はキャラクタアトリビュートの指定色になります。したがって、エスケープシーケンス等でキャラクタアトリビュートの色を変更すると、白以外の色が表示されます。また、ディップスイッチSW1–8をOFFにして利用すると、8色/4096色、16色/4096色、16色/1600万色、256色/1600万色のモードでの利用はできません。

INT CDH

ファンクション

No.4 描画プレーンの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	対象 VRAM
4H	WORD	描画プレーン

●対象 VRAM

システム VRAM の表示モードを設定する場合は 0、仮想 VRAM の表示モードを設定する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を設定します。

●描画プレーン

b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	p7	p6	p5	p4	p3	p2	p1	p0

b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}
ページ番号								0	0	0	0	0	0	0	0

ページはディスプレイに表示可能な画面の単位で、プレーンはページの構成要素です。

プレーンは描画したいプレーンに対応するビットを１に設定します。また、同時に複数のプレーンを指定することも可能です。ただし、指定できるプレーンはカラーモードによって次のようになります。

カラーモード	指定可能プレーン
モノクロ	p0～p2/p3/p7（*1）
8 色	p0～p2
16 色	p0～p3
256 色（*2）	p0～p7

（*1）モノクロモードの指定可能プレーンは機種によって異なります。詳しくはファンクション No.28（プレーン数の取得）を参照してください。

(*2) 256色/1600万色のカラーモードは256色オプションボード(PC–H98–E02)を実装した装置で専用高解像度版グラフィックスドライバを利用している場合のみ指定できます。

ページは0〜3の数で指定します。ただし、仮想VRAMに設定できるページは0のみです。システムVRAMに対して設定する場合は複数のページが利用できるので、表示するページと描画するページを別々に設定することもできます。設定できるページは解像度によって次のようになります。

解像度	ページ
640 × 200	0〜3 (*)
640 × 400	0〜1 (*)
1120 × 750	0

(*) 設定できるページは機能によっても異なります。詳しくはファンクションNo.28(プレーン数の取得)を参照してください。

ファンクションNo.3(表示モードの設定)を実行すると、描画プレーンは初期化されます。初期状態はカラーモードによって次のように異なります。

カラーモード	描画プレーン
モノクロ	ページ0、プレーン0
8色/8色、8色/4096色	ページ0、プレーン0〜2
16色/4096色、16色/1600万色	ページ0、プレーン0〜3
256色/1600万色	ページ0、プレーン0〜7

リターン

AX ＝0　**正常終了**

≠0　**異常終了**　(エラーコード一覧参照)

ハードウェア的に設定できないページ、プレーンを指定した場合は、異常終了になります。

解　説

システムVRAMまたは仮想VRAMの実際に描画を行うページ、プレーンの設定を行います。

INT CDH

ファンクション
No.5 表示プレーンの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	表示プレーン

表示プレーンの指定方法は、描画プレーンと同じです。ファンクション No.4（描画プレーンの設定）を参照してください。

リターン

AX ＝0　正常終了
　　≠0　異常終了　　（エラーコード一覧参照）
　　ハードウェア的に設定できないページ、プレーンを指定した場合は、異常終了になります。

解　説

システム VRAM の表示するページ、プレーンの設定を行います。

INT CDH

パレットの設定

コール

スタック=データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	パレット番号
8H	DWORD	カラーコード

●パレット番号

パレット番号はカラーモードによって次のようになります。

モノクロ

0〜1 で指定します。この範囲を超えたときは、2 の剰余が取られます。

8 色/8 色、8 色/4096 色

0〜7 で指定します。この範囲を超えたときは、8 の剰余が取られます。

16 色/4096 色、16 色/1600 万色

0〜15 で指定します。この範囲を超えたときは、16 の剰余が取られます。

256 色/1600 万色

0〜255 で指定します。この範囲を超えたときは、256 の剰余が取られます。

●カラーコード

b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
red								blue							

b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}
0	0	0	0	0	0	0	0	green							

green、red、blue は、緑、赤、青色の各階調を表します。
各カラーモードでの設定は、次のようになります。

モノクロ

green、red、blue の中で 1 つでも最上位ビットが 1 の場合は白色、それ以外は黒色になります。

8色/8色

green、red、blueのそれぞれの最上位ビットによって、次のように決まります。

green	red	blue	表示色
0	0	0	ブラック
0	0	1	ブルー
0	1	0	レッド
0	1	1	マゼンダ
1	0	0	グリーン
1	0	1	シアン
1	1	0	イエロー
1	1	1	ホワイト

8色/4096色、16色/4096色

green、red、blueの各バイトの上位4ビットだけ取り出して、それぞれ16階調とし、その組み合わせによって4096色を選択します。4ビットで表される0〜Fは値が大きいほど明るくなります。

各モードの初期値は、次のとおりです。

モノクロ

0	0 0 0 0 0 0
1	F F F F F F

8色/8色、8色/4096色

0	0 0 0 0 0 0
1	0 0 0 0 F F
2	0 0 F F 0 0
3	0 0 F F F F
4	F F 0 0 0 0
5	F F 0 0 F F
6	F F F F 0 0
7	F F F F F F

16色/4096色

0	0 0 0 0 0 0	8	7 7 7 7 7 7
1	0 0 0 0 F F	9	0 0 0 0 A A
2	0 0 F F 0 0	10	0 0 A A 0 0
3	0 0 F F F F	11	0 0 A A A A
4	F F 0 0 0 0	12	A A 0 0 0 0
5	F F 0 0 F F	13	A A 0 0 A A
6	F F F F 0 0	14	A A A A 0 0
7	F F F F F F	15	A A A A A A

16色/1600万色、256色/1600万色

green、red、blueの各階調によって1600万色のカラーコードを設定します。

16色/1600万色モードのパレットの初期値は、16色/4096色モードのパレットと同じです。256色/1600万色モードのパレットの初期値は、16色/4096色モードのパレットの初期値を16回繰り返したものになります。つまりパレット0〜15の色が、それぞれパレット16×n〜16×n+15（n＝1〜15）と同じ色になっています。

リターン AX ＝0 正常終了
≠0 異常終了 （エラーコード一覧参照）

解 説

指定されたパレット番号に対応するカラーコードを設定します。

グラフィックスドライバで色を表示するにはパレットを使用します。パレットはそれぞれのカラーモードで同時に使用できる色数だけ用意され、パレット番号0、パレット番号1…と番号がつけられています。それぞれのパレットにはカラーコード（実際に表示される色）があらかじめ設定されており、描画ファンクションでは表示色としてこのパレット番号を指定します。

このファンクションでは、このパレット番号とカラーコードの対応を変更することができます。すでに表示されているパレット番号を変更した場合には、表示色が新しく設定されたカラーコードに変わります。

注意 パレット番号を指定する場合、xx色/1600万色のカラーモードは、PC-H98で専用高解像度版グラフィックスドライバを利用している場合のみ指定できます。また、256色/1600万色のカラーモードは256色オプションボード（PC-98H-E02）を実装した装置でのみ指定できます。

INT CDH

ファンクション
No.7 ビューポート領域の設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	対象 VRAM
4H	WORD	ビューポート領域左上 X 座標
6H	WORD	ビューポート領域左上 Y 座標
8H	WORD	ビューポート領域右下 X 座標
0AH	WORD	ビューポート領域右下 Y 座標

●対象 VRAM

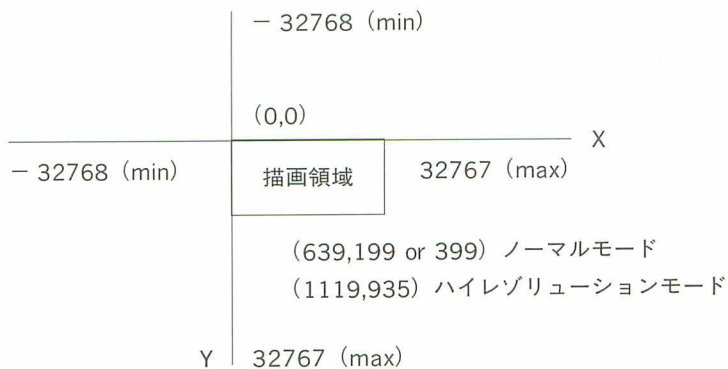
システム VRAM のビューポート領域を設定する場合は 0、仮想 VRAM のビューポート領域を想定する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

リターン

AX ＝0　正常終了
　≠0　異常終了　　（エラーコード一覧参照）

解　　説

システム VRAM または仮想 VRAM の実際に描画される領域（ビューポート領域）を設定します。

図形描画時に指定する座標はディスプレイ画面左上を原点とした、次の図のような整数系（−32768〜32767）座標ですが、実際に描画可能な領域は各解像度でディスプレイに表示できる範囲です。

各解像度で実際の描画可能な領域のことをビューポート領域といいます。このファンクションを実行することにより、ビューポート領域を変更することができます。

各解像度の最大ビューポート領域は次のとおりです。

・システム VRAM に設定する場合

解像度	最大ビューポート領域
640 × 200	(0,0) − (639,199)
640 × 400	(0,0) − (639,399)
1120 × 750	(0,0) − (1119,935)

・仮想 VRAM に設定する場合

最大ビューポート領域はファンクション No.2（仮想 VRAM の生成）を実行したときのビューポート領域になります。このファンクションでビューポート領域を変更しても、それ以前に描画されていたグラフィックは消去されません。

また、ビューポート領域はファンクション No.3（表示モードの設定）を実行すると各解像度の最大ビューポート領域になります。

No.7

INT CDH

ファンクション No.8 フォアグラウンドカラーの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	対象 VRAM
4H	DWORD	フォアグラウンドカラー（パレット番号）

●対象 VRAM

システム VRAM のフォアグラウンドカラーを設定する場合は 0、仮想 VRAM のフォアグラウンドカラーを設定する場合は、仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

●フォアグラウンドカラー

描画ファンクションで描画色を省略したときに用いられる色をパレット番号で指定します。パレット番号についてはファンクション No.6（パレット設定）を指定してください。

ファンクション No.3（表示モードの設定）を実行すると、カラーモードによって次のように初期化されます。

モノクロ　　：パレット番号 1
モノクロ以外　：パレット番号 7

リターン

AX ＝0　正常終了
　 ≠0　異常終了　（エラーコード一覧参照）

解　説

システム VRAM または仮想 VRAM のフォアグラウンドカラーの設定を行います。

INT CDH

ファンクション
No.9 バックグラウンドカラーの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	対象 VRAM
4H	DWORD	バックグラウンドカラー（パレット番号）

●対象 VRAM

システム VRAM のバックグラウンドカラーを設定する場合は 0、仮想 VRAM のバックグラウンドカラーを設定する場合は、仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

●バックグラウンドカラー

ファンクション No.14（画面消去）やファンクション No.26（領域移動）で用いられる色をパレット番号で指定します。ファンクション No.3（表示モードの設定）を実行すると、パレット番号 0 の色に初期化されます。

リターン

AX ＝0　正常終了
　　≠0　異常終了　　（エラーコード一覧参照）

解　説

システム VRAM または仮想 VRAM のバックグラウンドカラーの設定を行います。

INT CDH

ファンクション

No.10 ボーダーカラーの設定

コール

スタック=データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	ボーダーカラー

●ボーダーカラー

カラーモードによって次のように指定します。

モノクロモード

値	表示色
0	ブラック
1	ホワイト

8色/8色モード

値	表示色
0	ブラック
1	ブルー
2	レッド
3	マゼンダ
4	グリーン
5	シアン
6	イエロー
7	ホワイト

リターン

AX ＝0　正常終了
　≠0　異常終了　（エラーコード一覧参照）

解　説

ボーダーカラーの設定を行います。

このファンクションは、標準ディスプレイ（解像度640×200ドット）接続時のみ使用できます。ファンクションNo.3（表示モードの設定）を実行すると、0（ブラック）に初期化されます。

INT CDH

ファンクション No.11　表示スイッチの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
4H	BYTE	表示スイッチ

●表示スイッチ

表示スイッチ　＝0　非表示状態
　　　　　　　≠0　表示状態

リターン

AX ＝0　正常終了
　≠0　異常終了　　（エラーコード一覧参照）

解　説

システム VRAM のグラフィックをディスプレイに表示するか表示しないかを設定します。グラフィックの開始直後は、0（非表示状態）になっています。

INT CDH

ファンクション No.12 表示領域の設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
4H	WORD	システム VRAM 上の Y 座標

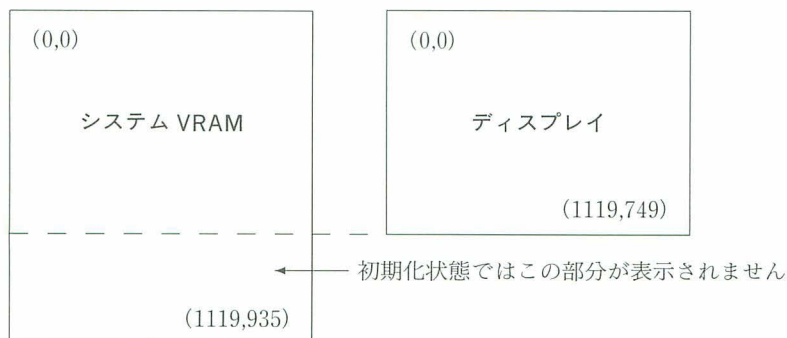
リターン

AX ＝0　正常終了
　≠0　異常終了　　(エラーコード一覧参照)

解　　説

システム VRAM 上の表示領域を設定します。このファンクションはハイレゾリューションモードでのみ使用可能です。

VRAM 上の Y 座標は、VRAM のどの位置からディスプレイに表示させるかを 0〜186 の範囲の値で指定します。

ハイレゾリューションモードでは、VRAM のサイズが実際にディスプレイに表示されるサイズより大きいため同時に VRAM の内容のすべてを見ることができません。ただし、このファンクションで表示領域を変えることで、VRAM 上のどの領域でも見ることができます。

初期化時は Y 座標 0 から 749 まで表示されます。

INT CDH

ファンクション No.13 中断処理ルーチンの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	中断処理ルーチンの先頭アドレス

リターン

AX ＝0　正常終了
　　≠0　異常終了　　（エラーコード一覧参照）

解　説

中断処理ルーチンのアドレスをグラフィックスドライバに通知します。

各ファンクション実行中に、何らかの事象が発生したときに特別な処理を行いたい場合（例えば STOP キーが押されたらファンクションの実行を中断する）には、この中断処理ルーチンの設定をあらかじめ実行します。

このファンクションを実行すると、グラフィックスドライバはファンクション No.22（閉領域の塗りつぶし）を実行中に、一定した処理ごとに指定された中断処理ルーチンをコールします。

したがって中断処理ルーチンには中断処理を行いたい事象の発生の検出（ STOP キーが押されたかなど）と、事象が発生した場合に行いたい特別な処理を記述しておきます。パラメータブロックにはこの中断処理ルーチンの先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。ただし、ファンクション No.0（グラフィックの開始）直後は、グラフィックスドライバ内の中断処理ルーチン（RET のみ）のアドレスが定義されています。この中断処理ルーチンのアドレスは、このファンクションで再設定されるまで有効になります。

注意　中断処理ルーチンを作成する場合は、次のことに注意してください。

- すべてレジスタを保障してください。
- グラフィックスドライバが管理するハードウェア（グラフィック VRAM、GDC、グラフィックチャージャ、EGC）の状態を変更しないでください。
- このルーチンの中で、グラフィックスドライバのファンクションをコールしないでください。
- グラフィックスドライバのデータ領域を変更しないでください。
- 中断処理ルーチンからリターンしない場合は、ユーザー側でスタックを解決してください。

INT CDH

ファンクション No.14 画面消去

コール

スタック=データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定する）

リターン

AX =0　正常終了
　≠0　異常終了　（エラーコード一覧参照）

解　説

システムVRAMに描画されている内容を消去します。

消去の対象になるのはビューポート領域のみです。このとき、バックグラウンドカラーに設定されている色で消去します。バックグラウンドカラーについてはファンクションNo.9（バックグラウンドカラーの設定）を参照してください。

INT CDH

点の描画

コ　ー　ル

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	BYTE	動作番号
8H	WORD	点を描画するX座標
0AH	WORD	点を描画するY座標
0CH	DWORD	点の色を表すパレット番号

●ラスタオペレーション（ROP）番号

次の16種類の中から指定します。

ROP番号	論理演算
0000	S
0001	$\overline{S}$
0002	D
0003	$\overline{D}$
0004	D＋S
0005	D＋$\overline{S}$
0006	$\overline{D}$＋S
0007	$\overline{D}$＋$\overline{S}$
0008	D・S
0009	D・$\overline{S}$
000A	$\overline{D}$・S
000B	$\overline{D}$・$\overline{S}$
000C	D（+）S
000D	D（+）$\overline{S}$
000E	$\overline{D}$（+）S
000F	$\overline{D}$（+）$\overline{S}$

D：VRAM上の現在のパレット番号
$\overline{D}$：Dを反転（NOT）したパレット番号
S：描画するパレット番号
$\overline{S}$：Sを反転（NOT）したパレット番号

＋　：OR（論理和）
・　：AND（論理積）
(＋)：XOR（排他的論理和）

●動作番号

01H＝点の色を省略した場合はフォアグラウンドカラーを使用
02H＝点の色を省略した場合はバックグラウンドカラーを使用

● X座標/Y座標

描画したい点を整数系（–32768～32767）座標で指定します。

●点の色を表すパレット番号

表示したい点の色をパレット番号で指定します。点の色を省略したいときは、最上位ビットに1を設定してください。

リターン

AX ＝0　**正常終了**
　　≠0　**異常終了**　　（エラーコード一覧参照）
　　動作番号に上記以外の値が設定された場合は異常終了になります。

解　　説

システムVRAM上の指定の座標に、指定のパレット番号で点を描画します。

ラスタオペレーションは色を表すパレットに対して働くので、すべての描画ファンクションで使用できます。

たとえば、カラーモードが16色/4096色で座標（100,100）、パレット番号が2のときに、この座標にパレット番号1で点を描画すると、ラスタオペレーション番号によって次のように描画後のパレット番号が異なります。

ラスタオペレーション番号	描画後のパレット番号
0	1
1	14
2	2
3	13
4	3
5	14
6	13
7	15
8	0
9	2
10	1
11	12
12	3
13	12
14	12
15	3

パレット番号1の反転（NOT）はモノクロの場合パレット番号0となり、8/8色、8/4096色の場合は6となります。

INT CDH

ファンクション No.16 線の描画

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	描画始点X座標
8H	WORD	描画始点Y座標
0AH	WORD	描画終点X座標
0CH	WORD	描画終点Y座標
0EH	BYTE	描画フラグ
10H	DWORD	線の色を表すパレット番号
14H	WORD	線幅
16H	BYTE	線種パターン長
18H	WORD	線種パターン
1AH	WORD	拡張線種パターン

●ラスタオペレーション番号

ファンクションNo.15（点の描画）を参照してください。

●X座標/Y座標

描画したい線の始点と終点を整数系（–32768〜32767）座標で指定します。

●描画フラグ

00H＝線種パターン、線幅省略（幅1ドットの実線）
01H＝線種パターンのみ指定（幅1ドット）
10H＝線幅のみ指定（実線）
11H＝線種パターン、線幅指定

●線の色

描画したい線の色をパレット番号で指定します。
最上位ビットに1を設定するとフォアグラウンドカラーで描画されます。

●線幅

0〜15の値で指定します。（線幅＋1）ドットの幅で線が描画されます。
線幅は、幅1ドットの線を中心として、描画する線が仰角45度以下の場合は

上下方向に、45度以上の場合は水平方向に描画されます。したがって、指定した線幅が奇数の場合には上下方向に描画される場合は下側、水平方向に描画される場合は右側がその反対側より1ドット分多くなります。

●線種パターン長

10H ＝ 16ビット（線種パターンを使用する）

20H ＝ 32ビット（線種パターンと拡張線種パターンを使用する）

●線種パターン、拡張線種パターン

破線などの線種をビットパターンで指定します。直線は指定されたビットパターンを繰り返し、ドットに対応して描画されます。このビットパターンは線種パターン長で16ビット単位か32ビット単位に指定できます。拡張線種パターンは32ビット単位のときに指定します。

リターン

AX ＝0　**正常終了**

　≠0　**異常終了**（エラーコード一覧参照）

解　説

システムVRAM上の指定した2点を結ぶ直線を、指定したパレット番号、線種パターン、線幅で描画します。

線種パターンとディスプレイ上のドットイメージの関係は次のようになります。

・16ビット単位で線種パターンを使用する場合

線種パターン長：10H

線種パターン　：0F0EH

ディスプレイ上のドットイメージ

○○○○●●●●○○○○●●●○

・32ビット単位で線種パターンを使用する場合

線種パターン長　：20H

線種パターン　　：0F0EH

拡張線種パターン：0C08H

ディスプレイ上のドットイメージ

○○○○●●●●○○○○●●●○○○○○●●○○○○○○●○○○

INT CDH

ファンクション No.17 三角形の描画

コール

スタックニデータ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	第1X座標
8H	WORD	第1Y座標
0AH	WORD	第2X座標
0CH	WORD	第2Y座標
0EH	WORD	第3X座標
10H	WORD	第3Y座標
12H	BYTE	描画フラグ
14H	DWORD	線の色を表すパレット番号
18H	WORD	線幅
1AH	BYTE	線種パターン長
1CH	WORD	線種パターン
1EH	WORD	拡張線種パターン
20H	BYTE	塗りつぶしフラグ
22H	DWORD	塗りつぶす色を表すパレット番号
22H	WORD	塗りつぶしに使用するタイルパターンの長さ
24H	DWORD	ダイルパターンの格納域のアドレス

●ラスタオペレーション番号

ファンクション No.15（点の描画）を参照してください。

● X座標/Y座標

描画したい三角形の各頂点を整数系（–32768～32767）座標で指定します。

●描画フラグ、線の色、線幅、線種パターン長、線種パターン、拡張線種パターン

ファンクション No.15（点の描画）を参照してください。

●塗りつぶしフラグ

00H ＝塗りつぶさない
01H ＝指定のパレット番号で塗りつぶす（省略時は、線の色で塗りつぶす）
02H ＝指定のタイルパターンで1バイト単位に塗りつぶす

03H ＝指定のタイルパターンで 2 バイト単位に塗りつぶす

●塗りつぶす色

塗りつぶす色をパレット番号で指定します。

パレット番号についてはファンクション No.6（パレットの設定）を参照してください。

●タイルパターン長

タイルパターン長は、モノクロのときは 1 以上、8 色/8 色、8 色/4096 色のときは 3 以上、16 色/4096 色のときは 4 以上の長さをバイト数で指定します。

●タイルパターン格納域

プレーン 0 から順番に塗りつぶすタイルパターンを、1 バイト単位（あるいは 2 バイト単位）で格納します。

パラメータブロックのオフセット 24H のダブルワードには、このタイルパターン格納域のアドレス（上位ワード＝セグメント、下位ワード＝オフセット）を格納します。

リターン

AX ＝0 **正常終了**

≠0 **異常終了** （エラーコード一覧参照）

解　　説

システム VRAM 上の指定した 3 点を結ぶ三角形を描画して必要であれば、内部を塗りつぶします。
タイルパターンはプレーン 0 から順番に各プレーンに対応しています。
たとえば、n+1 バイトのタイルパターンが次のように格納されているとします。

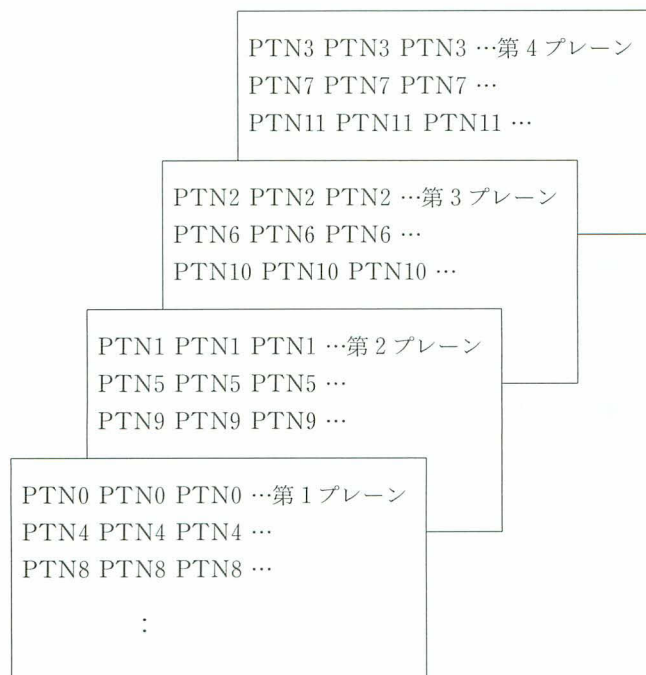
PTN0	PTN1	PTN2	…	PTNn

これを 16 色/4096 色で画面上に 1 バイト単位で展開させると次のようになります。

```
PTN3 PTN3 PTN3 …第4プレーン
PTN7 PTN7 PTN7 …
PTN11 PTN11 PTN11 …

PTN2 PTN2 PTN2 …第3プレーン
PTN6 PTN6 PTN6 …
PTN10 PTN10 PTN10 …

PTN1 PTN1 PTN1 …第2プレーン
PTN5 PTN5 PTN5 …
PTN9 PTN9 PTN9 …

PTN0 PTN0 PTN0 …第1プレーン
PTN4 PTN4 PTN4 …
PTN8 PTN8 PTN8 …
        :
```

塗りつぶしフラグの設定によって2バイト単位に展開させることもできます。

タイルパターンの長さが、条件より小さいときは処理は行いません。また、タイルパターンの長さと各表示モードのプレーン数が対応しない場合、余りは無視されます。

ラスタオペレーションで0以外の値を指定すると、頂点が正確に描画できないことがあります。

INT CDH

ファンクション

No.18 長方形の描画

コール

スタックデータ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	線分の開始X座標
8H	WORD	線分の開始Y座標
0AH	WORD	線分の終了X座標
0CH	WORD	線分の終了Y座標
0EH	BYTE	描画フラグ
10H	DWORD	線の色を表すパレット番号
14H	WORD	線幅
16H	BYTE	線種パターン長
18H	WORD	線種パターン
1AH	WORD	拡張線種パターン
1CH	BYTE	塗りつぶしフラグ
1EH	DWORD	塗りつぶす色を表すパレット番号
1EH	WORD	塗りつぶしに使用するタイルパターンの長さ
20H	DWORD	タイルパターンの格納域のアドレス

●ラスタオペレーション番号

ファンクション No.15（点の描画）を参照してください。

● X座標/Y座標

描画したい長方形の対角線の始点、終点を整数系（–32768～32767）座標で指定します。

●描画フラグ、線の色、線幅、線種パターン長、線種パターン、拡張線種パターン

ファンクション No.16（線の描画）を参照してください。

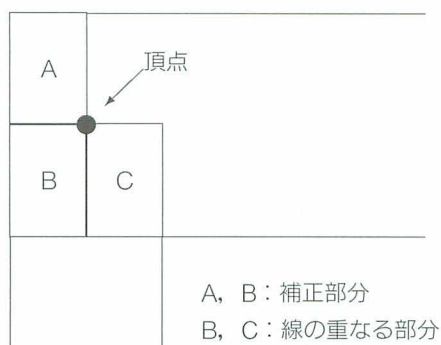
●塗りつぶしフラグ、塗りつぶす色、タイルパターンの長さ、タイルパターン格納域

ファンクション No.17（三角形の描画）を参照してください。

リターン　　AX ＝0　正常終了
　　　　　　　　≠0　異常終了　　（エラーコード一覧参照）

解　　説

システム VRAM 上の指定された 2 点を結ぶ線分を対角線とする長方形を描画し、必要であれば内部を塗りつぶします。

線幅を指定する場合は、各頂点で水平方向の線による補正を行うため、頂点は次のようになります。

注意　ラスタオペレーションで 0 以外の値を指定すると、頂点が正確に描画されないことがあります。

INT CDH

ファンクション No.19 台形の描画

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定）
4H	WORD	ラスタオペレーション番号
6H	WORD	第1X座標
8H	WORD	第1Y座標
0AH	WORD	第2X座標
0CH	WORD	第3X座標
0EH	WORD	第3Y座標
10H	WORD	第4X座標
12H	BYTE	描画フラグ
14H	DWORD	線の色を表すパレット番号
18H	WORD	線幅
1AH	BYTE	線種パターン長
1CH	WORD	線種パターン
1EH	WORD	拡張線種パターン
20H	BYTE	塗りつぶしフラグ
22H	DWORD	塗りつぶす色を表すパレット番号
22H	WORD	塗りつぶしに使用するタイルパターンの長さ
24H	DWORD	タイルパターンの格納域のアドレス

●ラスタオペレーション番号

ファンクション No.15（点の描画）を参照してください。

● X座標/Y座標

描画したい台形の始点、終点を整数系（–32768〜32767）座標で指定します。

●描画フラグ、線の色、線幅、線種パターン長、線種パターン、拡張線種パターン

ファンクション No.16（線の描画）を参照してください。

●塗りつぶしフラグ、塗りつぶす色、タイルパターンの長さ、タイルパターン格納域

ファンクション No.17（三角形の描画）を参照してください。

リターン　AX ＝０　正常終了
　≠０　異常終了　（エラーコード一覧参照）

解　説

システムVRAM上の指定された４点を頂点とする台形を描画し、必要であれば内部を塗りつぶします。

注意　ラスタオペレーションで０以外の値を指定すると、頂点が正確に描画されないことがあります。

INT CDH

ファンクション No.20 円の描画

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	中心点X座標
8H	WORD	中心点Y座標
0AH	WORD	半径
0CH	WORD	開始点X座標
0EH	WORD	開始点Y座標
10H	WORD	終了点X座標
12H	WORD	終了点Y座標
14H	BYTE	描画フラグ
16H	DWORD	線の色を表すパレット番号
1AH	BYTE	形状フラグ
1CH	BYTE	線種パターン長
1EH	WORD	線種パターン
20H	WORD	拡張線種パターン
22H	BYTE	塗りつぶしフラグ
24H	DWORD	塗りつぶす色を表すパレット番号
24H	WORD	塗りつぶしに使用するタイルパターンの長さ
26H	DWORD	タイルパターンの格納域のアドレス

●ラスタオペレーション番号

ファンクション No.15（点の描画）を参照してください。

●中心点座標、半径、開始点座標、終了点座標

描画したい円の中心点と半径を整数系（–32768～32767）座標で指定します。円弧、扇型を描画する場合は、さらに開始点座標と終了点座標を指定します。

●描画フラグ

00H ＝線種パターン省略（実線）
01H ＝線種パターン指定

●形状フラグ

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	×	×	×	×	×

b_0　＝開始座標フラグ
　0　開始座標指定なし
　1　開始座標指定あり
b_1　＝開始線分フラグ
　0　開始点と中心点間に線分を描画しない
　1　開始点と中心点間に線分を描画する
b_2　＝終了座標フラグ
　0　終了座標指定なし
　1　終了座標指定あり
b_3　＝終了線分フラグ
　0　終了点と中心点間に線分を描画しない
　1　終了点と中心点間に線分を描画する
b_4　＝描画範囲フラグ
　0　開始点、終了点が等しい場合は、全円を描画する
　1　開始点、終了点が等しい場合は、開始点（終了点）を描画する

●線の色、線種パターン長、線種パターン、拡張線種パターン
ファンクション No.16（線の描画）を参照してください。

●塗りつぶしフラグ、塗りつぶす色、タイルパターンの長さ、タイルパターン格納域
ファンクション No.17（三角形の描画）を参照してください。

リターン

AX ＝0　正常終了
　≠0　異常終了　　（エラーコード一覧参照）

解　　説

システム VRAM 上の指定された中心点と半径をもとに円を描画します。また開始点座標、終了点座標を指定することにより円弧、扇形の描画も可能です。

注意　ラスタオペレーションで 0 以外の値を指定すると、頂点が正確に描画されないことがあります。

INT CDH

ファンクション
No.21　楕円の描画

コール　スタックニデータ領域の先頭アドレス

パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定してください）
4H	WORD	ラスタオペレーション番号
6H	WORD	中心点X座標
8H	WORD	中心点Y座標
0AH	WORD	X方向半径
0CH	WORD	Y方向半径
0EH	WORD	開始点X座標
10H	WORD	開始点Y座標
12H	WORD	終了点X座標
14H	WORD	終了点Y座標
16H	BYTE	描画フラグ
18H	DWORD	線の色を表すパレット番号
1CH	BYTE	形状フラグ
1EH	BYTE	線種パターン長
20H	WORD	線種パターン
22H	WORD	拡張線種パターン
24H	BYTE	塗りつぶしフラグ
26H	DWORD	塗りつぶす色を表すパレット番号
26H	WORD	塗りつぶしに使用するタイルパターンの長さ
28H	DWORD	タイルパターンの格納域のアドレス

●ラスタオペレーション番号

ファンクションNo.15（点の描画）を参照してください。

●X方向半径/Y方向半径

描画したい楕円の中心点からのX方向、Y方向の半径を整数系（–32768〜32767）で指定します。

●中心点座標、開始点座標、終了点座標

描画したい楕円の中心点を整数系（–32768〜32767）座標で指定します。楕円弧を描画する場合は、さらに開始点座標と終了点座標を指定します。

●描画フラグ、形状フラグ
ファンクション No.20（円の描画）を参照してください。

●線の色、線種パターン長、線種パターン、拡張線種パターン
ファンクション No.16（線の描画）を参照してください。

●塗りつぶしフラグ、塗りつぶす色、タイルパターンの長さ、タイルパターン格納域
ファンクション No.17（三角形の描画）を参照してください。

リターン

AX ＝0　正常終了
　≠0　異常終了　（エラーコード一覧参照）

解　説

システム VRAM 上の指定された中心点と X 方向半径および Y 方向半径をもとに、楕円を描画します。また、開始点座標、終了点座標を指定することにより楕円弧の描画もできます。

INT CDH

閉領域の塗りつぶし

コール　スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	描画開始X座標
8H	WORD	描画開始Y座標
0AH	DWORD	境界色を表すパレット番号
0EH	BYTE	塗りつぶしフラグ
10H	DWORD	塗りつぶす色を表すパレット番号
10H	WORD	塗りつぶしに使用するタイルパターンの長さ
12H	DWORD	タイルパターンの格納域のアドレス
16H	DWORD	作業域先頭アドレス
1AH	WORD	作業域の大きさ（バイト数）

●ラスタオペレーション番号

ファンクション No.15（点の描画）を参照してください。

●描画開始座標

塗りつぶしたい閉領域内の任意の点を整数系（–32768〜32767）座標で指定します。

●境界色

塗りつぶしたい領域の境界線の色をパレット番号で指定します。最上位ビットに1を設定するとフォアグラウンドカラーを採用します。

●塗りつぶしフラグ、塗りつぶす色、タイルパターンの長さ、タイルパターン格納域

ファンクション No.17（三角形の描画）を参照してください。

●作業域、作業域の大きさ

作業域はあらかじめ確保しておき、そのアドレス（上位ワード＝セグメント、下位ワード＝オフセット）と、その大きさをバイト数で指定します。

リターン　AX ＝0　正常終了
　　　　≠0　異常終了　　（エラーコード一覧参照）

解　　説

システム VRAM 上の指定された座標を含み、指定された境界色で囲まれる閉領域を塗りつぶします。

作業域の大きさは 16 バイト以上必要で、塗りつぶす閉領域の形状によって異なります。作業域が不足した場合はエラーコード 2 を返し処理を中断します。この場合は作業域を大きくとるか、分割して塗りつぶしてください。

ビューポート領域外に開始座標を指定した場合は何も描画されません。

INT CDH

ファンクション
No.23 グラフィックイメージの取得

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	左上X座標
6H	WORD	左上Y座標
8H	WORD	右下X座標
0AH	WORD	右下Y座標
0CH	WORD	メモリ上の格納域の長さ
0EH	DWORD	メモリ上の格納域のアドレス

●左上、右下座標

グラフィックイメージを取得したい矩形の左上と右下の頂点を整数系（–32768～32767）座標で指定します。

●格納域の長さ

メモリ上にあらかじめ確保しておいた格納域の長さをバイト数で指定します。ただし指定された矩形領域が（x1,y1）–（x2,y2）の場合は、次の条件を満たしていなければなりません。

格納域長≧（(x2–x1＋8）￥8）×（y2–y1＋1）×A＋4

ここで￥は割算の商の整数部をとることを意味します。またAの値はカラーモードによって次のようになります。

・モノクロ　：1
・8色/8色、8色/4096色　：3
・16色/4096色　：4

●格納域のアドレス

メモリ上の格納域のアドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。矩形領域を取得すると格納域には次の形式で格納されます。

オフセット	内容	
0	X 方向ドット数	Y 方向ドット数
4	Y 方向 1 ドット分の X 方向ドットパターン (1)	
4＋α	Y 方向 1 ドット分の X 方向ドットパターン (2)	
4＋2α	Y 方向 1 ドット分の X 方向ドットパターン (3)	
4＋3α	Y 方向 1 ドット分の X 方向ドットパターン (4)	
4＋Aα	以下太ワク部分を (Y 方向ドット数－1) 回分繰り返します。	
4＋βα		

メモリ内下位バイトから順に各ビットがX座標のドット（昇順）と対応します。

カラーモードがモノクロの場合 (2)、(3)、(4) 部分は存在しません。

8色のカラーモードの場合は、(4) のみ存在しません。

α＝（x2–x1＋8）¥8
β＝（y2–y1＋1）×A

カラーモードがカラーの場合の各ドットのパレット番号は、対応する X 方向ドットパターン（1）～（3）（16 色モードの場合は（1）～（4））のビット値（0/1）に、それぞれ 1、2、4（16 色モードの場合はさらに 8）を乗算し総和を取ったものです。カラーモードがモノクロの場合の各ドットの白／黒は、対応する X 方向ドットパターン（1）内のビット値が 1/0 で表されます。

リターン

AX ＝0　正常終了
　　≠0　異常終了　　（エラーコード一覧参照）

解　　説

システム VRAM 上の指定された 2 点を結ぶ線分を対角線とする、矩形領域のイメージを指定されたメモリ上の格納域に格納します。

格納域の長さが条件を満たしていない場合は、異常終了となり処理は行われません。一部分がビューポート領域からはみ出す場合は、はみ出した部分はバックグラウンドカラーと見なします。また、64K バイトを超えるグラフィックイメージの取得はできません。カラーモードがモノクロの場合は、描画プレーンのプレーン 0 からプレーン 3 の順で、マスクされていない最初のプレーンのグラフィックイメージを格納します。

注意　PC–H98 で専用高解像度版グラフィックスドライバを利用する場合は、次の点に注意してください。

- 256 色/1600 万色モードでは、プレーン数によりグラフィックイメージの格納形式における Y 方向 1 ドット分の X 方向ドットパターンが 8 個に拡張されます。
- 16 色/1600 万色モードは、16 色/4096 色モードのグラフィックイメージの格納形式と同じです。

INT CDH

No.24 グラフィックイメージの設定

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	左上X座標
8H	WORD	左上Y座標
0AH	WORD	メモリ上の格納域の長さ
0CH	DWORD	メモリ上の格納域のアドレス
10H	BYTE	カラースイッチ
12H	DWORD	フォアグラウンドカラー
16H	DWORD	バックグラウンドカラー

●ラスタオペレーション番号

ファンクション No.15（点の描画）を参照してください。

●左上座標

取得されたグラフィックイメージを、システム VRAM に展開する際の短形の左上頂点を整数系（–32768〜32767）座標で指定します。

●カラースイッチ

00H ＝フォアグラウンドカラー、バックグラウンドカラーの指定なし
（メモリ上のグラフィックイメージは、設定される側の画面モードでグラフィックイメージを取得した場合の格納形式であるものとして設定されます。）

01H ＝フォアグラウンドカラー、バックグラウンドカラーの指定あり
（メモリ上のグラフィックイメージは、モノクロモードでグラフィックイメージを取得した場合の格納形式であるものとして設定されます。）

●フォアグラウンドカラー

モノクロモードで格納されているグラフィックイメージの白（1）のドットを描画するパレット番号を指定します。

●バックグラウンドカラー

モノクロモードで格納されているグラフィックイメージの黒（0）のドットを描画するパレット番号を指定します。

リターン

AX ＝0　正常終了
　　≠0　異常終了　　（エラーコード一覧参照）

解　説

メモリ上を格納域に取得されたイメージを、システム VRAM 上の指定された１点を左上とする矩形領域上に展開します。ただし、ビューポート領域からはみ出す部分は描画されないので注意してください。

表示モードがカラーの場合は、描画プレーンのプレーン０からプレーン３の順で、マスクされていないプレーンにグラフィックイメージを展開します。したがって、グラフィックイメージを取得したときとグラフィックイメージを設定する場合の表示モード、描画プレーンの情報が異なる場合は、取得前のグラフィックイメージと設定後のグラフィックイメージが異なる可能性があります。

表示モードがモノクロの場合は、描画プレーンのプレーン０からプレーン３の順でマスクされていない最初のプレーンにグラフィックイメージを展開します。

INT CDH

領域転送

コール　スタック＝データ領域の先頭アドレス

パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	転送元のVRAM
4H	WORD	転送元の矩形領域の左上X座標
6H	WORD	転送元の矩形領域の左上Y座標
8H	WORD	転送元の矩形領域の右下X座標
0AH	WORD	転送元の矩形領域の右下Y座標
0CH	DWORD	転送先のVRAM
10H	WORD	ラスタオペレーション番号
12H	WORD	転送先のX座標
14H	WORD	転送先のY座標
16H	WORD	X方向倍率
18H	WORD	Y方向倍率
1AH	BYTE	裏返し
1BH	BYTE	回転

●転送元のVRAM、転送先のVRAM

システムVRAMを対象とする場合は0、仮想VRAMを対象とする場合は仮想VRAM構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

システムVRAMと仮想VRAMの相互の領域転送も可能です。

●転送元座標、転送先座標

グラフィックイメージの転送元の矩形の左上、右下頂点、および転送先の座標を整数系（–32768〜32767）座標で指定します。

●ラスタオペレーション番号

ファンクションNo.15（点の描画）を参照してください。

●X方向倍率、Y方向倍率

X方向、Y方向の各倍率は次のようになります。

拡　　大	縮　　小
0000H ＝等倍	8000H ＝等倍
0001H ＝ 2倍	8001H ＝ 1/2倍
0002H ＝ 4倍	8002H ＝ 1/4倍
0003H ＝ 8倍	8003H ＝ 1/8倍
0004H ＝ 16倍	8004H ＝ 1/16倍

●裏返し

転送する際に裏返すかどうかを指定します。裏返しは転送先の座標を含むX軸を、平行な線分を対象軸として線対象に描画されます。

00H ＝裏返しを行わない
00H ≠裏返しを行う

●回転

転送する際に回転するかどうかを指定します。回転は転送先の座標を原点にして、反時計回りに90゜単位で行います。

00H ＝回転を行わない
01H ＝ 90゜ 回転
02H ＝ 180゜ 回転
03H ＝ 270゜ 回転

リターン　AX ＝0　正常終了
　≠0　異常終了　（エラーコード一覧参照）

解　　説

指定された2点を結ぶ線文を結ぶ線分を対角線とする矩形領域の内容を、指定された1点を左上とする矩形領域に転送します。これはシステムVRAM、仮想VRAMのどちらでも転送できます。

注意　領域転送を行う場合、次のことに注意してください。

- 転送先のビューポート領域からはみ出す部分は描画されません。
- 転送元のビューポート領域からはみ出す部分は、バックグラウンドカラーと見なされます。
- 転送先の描画プレーンに対応する転送元のプレーンが転送されます。このとき、転送元のプレーンが描画プレーンでない場合は0が転送されます。
- 裏返しと回転を同時に指定した場合は、裏返しを先に行います。
- 転送元と転送先の矩形領域が同じVRAM内で重なる場合は、拡大、縮小、裏返し、回転を指定した場合の動作は保証できません。

INT CDH

ファンクション
No.26 領域移動

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	予約パラメータ（必ず0を指定する）
4H	WORD	ラスタオペレーション番号
6H	WORD	X方向移動ドット数
8H	WORD	Y方向移動ドット数
0AH	BYTE	クリアフラグ

●ラスタオペレーション番号
ファンクション No.15（点の描画）を参照してください。

● X方向移動ドット数
この値が正のときは左方向に移動し、負の場合は右方向に移動します。

● Y方向移動ドット数
この値が正のときは上方向に移動し、負の場合は下方向に移動します。

●クリアフラグ
移動によって、画面に新たに表示される領域を塗りつぶす色を指定します。

クリアフラグ ＝0　パレット番号0で塗りつぶす
クリアフラグ ≠0　バックグラウンドカラーで塗りつぶす

リターン

AX ＝0　**正常終了**
≠0　**異常終了**　（エラーコード一覧参照）

解　説

指定されたドット数分、画面（システムVRAM）上のデータを移動します。
移動の対象となるのはビューポート領域内です。

INT CDH

ファンクション
No.27 バージョンの取得

コール　スタック=データ領域の先頭アドレス

リターン　AX =バージョン番号
　AL =バージョン番号の整数部
　AH =バージョン番号の小数部

解　説

グラフィックスドライバのバージョンを取得します。
たとえば、バージョン1.0では、AXに0001Hが入ります。

注意　PC-H98で専用高解像度版グラフィックスドライバを利用する場合、このファンクションでAX = 0201Hが返されることを確認してください。

INT CDH

ファンクション
No.28 プレーン数の取得

コール　スタック＝データ領域の先頭アドレス

リターン　AH ＝バンク数
AL ＝プレーン数

解　説

プレーン数を取得します。

このファンクションによって、現在の環境で設定できる表示モード、描画プレーン、表示プレーンは次のようになります。

なお表中の「指定可能表示モード」の値は、ファンクション No.3（表示モードの設定）の表示モードとして指定可能な値です。また「指定可能ページ」はシステム VRAM に対する値です。

仮想 VRAM に対して指定できるのはページ 0 のみです。

取得バンク数	取得プレーン数	ハードウェア	指定可能表示モード	指定可能ページ	指定可能プレーン
1	3	ノーマルモード	0000H	0〜1	0〜2
			0001H	0〜1	0〜2
			0002H	0〜1	0〜2
			0100H	0	0〜2
			0101H	0	0〜2
			0102H	0	0〜2
	4	ノーマルモード	0000H	0〜1	0〜3
			0001H	0〜1	0〜2
			0002H	0〜1	0〜2
			0003H	0〜1	0〜3
			0100H	0	0〜3
			0101H	0	0〜2
			0102H	0	0〜2
			0103H	0	0〜3

取得バンク数	取得プレーン数	ハードウェア	指定可能表示モード	指定可能ページ	指定可能プレーン
1	4	ハイレゾリューションモード	0300H	0	0～3
			0303H	0	0～3
			0304H	0	0～3
	8	ハイレゾリューションモード	0300H	0	0～7
			0303H	0	0～3
			0304H	0	0～3
			0305H	0	0～7
2	3	ノーマルモード	0000H	0～3	0～2
			0001H	0～3	0～2
			0002H	0～3	0～2
			0100H	0～1	0～2
			0101H	0～1	0～2
			0102H	0～1	0～2
	4	ノーマルモード	0000H	0～3	0～3
			0001H	0～3	0～2
			0002H	0～3	0～2
			0003H	0～3	0～3
			0100H	0～1	0～3
			0101H	0～1	0～2
			0102H	0～1	0～2
			0103H	0～1	0～3
			0104H	0～1	0～3
	8	ノーマルモード	0000H	0～3	0～3
			0001H	0～3	0～2
			0002H	0～3	0～2
			0003H	0～3	0～3
			0100H	0～1	0～7
			0101H	0～1	0～2
			0102H	0～1	0～2
			0103H	0～1	0～3
			0104H	0～1	0～3
			0105H	0～1	0～7

注意　256色オプションボード（PC–H98–E02）を実装したPC–H98上で、専用高解像度版グラフィックスドライバを利用している場合のみプレーン数に8が返されます。

INT CDH

ファンクション No.29 表示モードの取得

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM の表示モードを取得する場合は 0、仮想 VRAM の表示モードを取得する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

リターン

パラメータブロック

オフセット	サイズ	内　容
4H	WORD	表示モード

AX ＝0　常に正常終了

解　説

システム VRAM または仮想 VRAM の現在の表示モードを取得します。
表示モードについては、ファンクション No.3（表示モードの設定）を参照してください。

INT CDH

ファンクション No.30 描画プレーンの取得

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	対象 VRAM

●対象 VRAM
　システム VRAM の描画プレーンを取得する場合は 0、仮想 VRAM の描画プレーンを取得する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

リターン

パラメータブロック

オフセット	サイズ	内　　容
4H	DWORD	描画プレーン

AX ＝ 0　常に正常終了

解　　説

システム VRAM または仮想 VRAM の現在の描画プレーンを取得します。
描画プレーンについては、ファンクション No.4（描画プレーンの設定）を参照してください。

INT CDH

ファンクション No.31 表示プレーンの取得

コール　スタック＝データ領域の先頭アドレス

リターン　パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	表示プレーン

AX ＝0　常に正常終了

解　説

現在の表示プレーンを取得します。
表示プレーンについては、ファンクション No.5（表示プレーンの設定）を参照してください。

INT CDH

ファンクション No.32 パレットの取得

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
4H	DWORD	パレット番号

リターン

パラメータブロック

オフセット	サイズ	内　　容
4H	DWORD	カラーコード

AX ＝0　常に正常終了

解　　説

指定されたパレットに設定されているカラーコードを取得します。
パレット番号、カラーコードについては、ファンクション No.6（パレットの設定）を参照してください。

INT CDH

ファンクション No.33 ビューポート領域の取得

コール

スタック=データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	対象 VRAM

●対象 VRAM
　システム VRAM のビューポート領域を取得する場合は 0、仮想 VRAM のビューポート領域を取得する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード=セグメント、下位ワード=オフセット）を指定します。

リターン

パラメータブロック

オフセット	サイズ	内　容
4H	WORD	ビューポート領域左上 X 座標
6H	WORD	ビューポート領域左上 Y 座標
8H	WORD	ビューポート領域右下 X 座標
0AH	WORD	ビューポート領域右下 Y 座標

AX ＝0　常に正常終了

解　説

システム VRAM または仮想 VRAM に現在設定されているビューポート領域の、左上点と右下点の座標を取得します。

ビューポート領域については、ファンクション No.7（ビューポート領域の設定）を参照してください。

INT CDH

ファンクション No.34 フォアグラウンドカラーの取得

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM のフォアグラウンドカラーを取得する場合は 0、仮想 VRAM のフォアグラウンドカラーを取得する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

リターン

パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	フォアグラウンドカラー

AX ＝0　常に正常終了

解　説

システム VRAM または仮想 VRAM に現在設定されているフォアグラウンドカラーのパレット番号を取得します。

フォアグラウンドカラーについては、ファンクション No.8（フォアグラウンドカラーの設定）を参照してください。

INT CDH

ファンクション No.35 バックグラウンドカラーの取得

コール

スタック＝データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM のバックグラウンドカラーを取得する場合は 0、仮想 VRAM のバックグラウンドカラーを取得する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

リターン

パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	バックグラウンドカラー

AX ＝0 常に正常終了

解 説

システム VRAM または仮想 VRAM に現在設定されているバックグラウンドカラーのパレット番号を取得します。

バックグラウンドカラーについては、ファンクション No.9（バックグラウンドカラーの設定）を参照してください。

INT CDH

ファンクション No.36 ボーダーカラーの取得

コール　スタック＝データ領域の先頭アドレス

リターン　パラメータブロック

オフセット	サイズ	内　　容
4H	DWORD	ボーダーカラー

AX ＝０　常に正常終了

解　　説

ボーダーカラーの現在の設定値を取得します。
ボーダーカラーについては、ファンクション No.10（ボーダーカラーの設定）を参照してください。

INT CDH

ファンクション No.37 表示スイッチの取得

コール スタック＝データ領域の先頭アドレス

リターン パラメータブロック

オフセット	サイズ	内　容
4H	BYTE	表示スイッチ

AX ＝0　常に正常終了

解　説

表示スイッチの現在の設定状況を取得します。
表示スイッチについては、ファンクション No.11（表示スイッチの設定）を参照してください。

INT CDH

ファンクション No.38

指定座標のパレットの取得

コール

スタックニデータ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
0H	DWORD	対象 VRAM
4H	WORD	取得したい点の X 座標
6H	WORD	取得したい点の Y 座標

●対象 VRAM

システム VRAM の座標のパレット番号を取得する場合は 0、仮想 VRAM の座標のパレット番号を取得する場合は仮想 VRAM 構造体の先頭アドレス（上位ワード＝セグメント、下位ワード＝オフセット）を指定します。

● X 座標、Y 座標

パレット番号を取得したい点を整数系（–32768～32767）座標で指定します。

リターン

パラメータブロック

オフセット	サイズ	内　　容
4H	DWORD	点のパレット番号

AX ＝ 0　常に正常終了

解　　説

システム VRAM または仮想 VRAM の指定された座標上にある、点のパレット番号を取得します。

指定の座標がビューポート領域外であれば、点のパレット番号として–1（FFFFFFFFH）を取得します。

INT CDH

ファンクション No.39 表示領域の取得

コール　スタック＝データ領域の先頭アドレス

リターン　パラメータブロック

オフセット	サイズ	内　容
4H	WORD	システム VRAM 上の Y 座標

AX ＝0　常に正常終了

解　説

システム VRAM 上の表示領域を取得します。

このファンクションは、ハイレゾリューションモードでのみ意味を持ち、VRAM 上の表示開始 Y 座標を取得します。

表示領域については、ファンクション No.12（表示領域の設定）を参照してください。なお、ノーマルモードでは常に 0 を取得します。

INT CDH

ファンクション
No.40 中断処理ルーチンの取得

コール　スタック＝データ領域の先頭アドレス

リターン　パラメータブロック

オフセット	サイズ	内　容
4H	DWORD	中断処理ルーチンのアドレス

AX ＝0　常に正常終了

解　説

現在、設定されている中断処理ルーチンの先頭アドレスを取得します。アドレスは上位ワードがセグメント、下位ワードがオフセットです。

中断処理ルーチンについてはファンクションNo.13（中断処理ルーチンの設定）を参照してください。

4.4 エラーコード一覧

各ファンクションではリターン値がAX ≠ 0のときは、エラーコードを返します。エラーコードとその意味については次のとおりです。

エラーコード	意　味
01	不正呼び出し。パラメータに誤りがあります。
02	作業域不足。上位から受け取った作業域のサイズでは、機能が実行できません。
03	演算結果のオーバフロー。楕円描画などで不正なパラメータにより描画の実行が不可能です。
04	不正ファンクション。現在このファンクションは定義されていません。
05(*)	データ領域のアドレス不正。EMS利用時に不正なアドレスがパラメータに指定されているため、機能の実行が不可能です。EMS領域内のアドレスが指定された場合などにエラーとなります。

(*)PC–H98で専用高解像度版グラフィックスドライバを使用している場合のみ返されるものです。

4.5 専用高解像度版(GRP_H98.LIB)とGRAPH.LIBの互換性について

PC–H98で専用高解像度版グラフィックスドライバ（GRP_H98.LIB）を組み込んだ場合、使用するグラフィックス用ハードウェアの特性により、一部の描画ファンクションで描画結果が非互換となります。詳細は次の表を参照してください。

描画ファンクション	描画結果がGRAPH.LIBと異なる点	条件	備考
三角形の描画	輪郭を構成する点の位置が1ドットずれることがある。	線種パターンと線幅を省略し、輪郭と同じ色で内部を塗りつぶす場合。	線種パターンを32ビットで指定すると描画結果、処理速度はGRAPH.LIBと同じになる。
台形の描画	同上	線種パターンと線幅を省略し、輪郭と同じ色で内部を塗りつぶす場合。	
円の描画	扇形の終了線分を構成する点の位置が1ドットずれることがある。またラスタオペレーションで0以外の値を指定すると、扇形の弧と開始線分と終了線分の交点で描画色が異なる。	線種のパターンを省略し、塗りつぶしを行わない扇形を描画した場合。	

描画ファンクション	描画結果が GRAPH.LIB と異なる点	条 件	備 考
楕円の描画	扇形の終了線分を構成する点の位置が 1 ドットずれることがある。またラスタオペレーションで 0 以外の値を指定すると、扇形の弧と開始線分と終了線分の交点で描画色が異なる。	線種パターンを省略し、楕円、弧、扇形、輪郭と同じ色で内部を塗りつぶす楕円を描画した場合。	線種パターンを 32 ビットで指定すると描画結果、処理速度は GRAPH.LIB と同じになる。
領域転送	縮小するときはドットの間引き方によって転送結果が異なる。また、拡大するときは転送開始位置が（拡大率－ 1）ドット異 なる。	転送元と転送先が共にシステム VRAM であり、拡大または縮小を行う場合。	仮想 VRAM を経由してシステム VRAM に転送すると、描画結果は GRAPH.LIB と 同じになる。ただし処理速度は低下する。

4.6　プログラム例

ここではマクロアセンブラ（MASM）と C 言語（MS–C）のプログラム例を掲載します。ただし C 言語の場合、MS–C のバージョン 3.0 でコンパイルするときは、オプション（/Ze）を指定します。

■ マクロアセンブラでの使用例

```
GINITIAL        EQU     00*4                        ; グラフ開始    :
                                                    ファンクション No. 0
GTERM           EQU     01*4                        ; グラフ終了    :
                                                    ファンクション No. 1
GEDSPSW         EQU     11*4                        ; 表示スイッチ  :
                                                    ファンクション No.11
GDPSET          EQU     15*4                        ; 点描画        :
                                                    ファンクション No.15

; データエリア

DATA          SEGMENT
GDDEV_NAME    DB       'GRAPH    $'                 ; グラフィックスドラ
              DB       0                            ; イバのデバイス名
GRAPH_ADDR    DD       0                            ; エントリテーブルの
                                                    ; 先頭アドレス
PARA_AREA     DB       2048  DUP(0)                 ; パラメータブロック
```

```
                                                        ; とワークエリア
DATA            ENDS

; スタック

STACK           SEGMENT         STACK
                DW      256     DUP(0)
STACK           ENDS

; コード部

CODE            SEGMENT
                ASSUME CS : CODE,DS : DATA
START :         MOV     AX,DATA
                MOV     DS,AX
                CALL    DRV_CHK                         ; ドライバの存在を
                                                        ; 確認する
                JNC     SMPL_START
                JMP     SMPL_END
SMPL_START :    CALL    GETGD_ENT                       ; エントリアドレス
                                                        ; の取得
                MOV     SI,GINITIAL                     ; グラフィックス
                                                        ; の開始
                CALL    FCALL
                MOV     BX,OFFSET DS : PARA_AREA        ; 点の描画
                MOV     WORD PRT DS : [BX],0            ; 予約パラメータ
                MOV     WORD PRT DS : 2[BX],0
                MOV     WORD PTR DS : 4[BX],0           ; ラスタオペレー
                                                        ; ション番号
                MOV     WORD PTR DS : 6[BX],1           ; 動作番号
                MOV     WORD PTR DS : 8[BX],500         ; 点を描画する X 座標
                MOV     WORD PTR DS : 0AH[BX],100       ; 点を描画する Y 座標
                MOV     WORD PTR DS : 0CH[BX],7         ; パレット番号
                                                        ; (下位 16BITS)
                MOV     WORD PTR DS : 0EH[BX],0         ; パレット番号
                                                        ; (上位 16BITS)
                MOV     SI,GDPSET
                CALL    FCALL
                MOV     BX,OFFSET DS : PARA_AREA        ; 表示スイッチの設定
                MOV     WORD PTR DS : 4[BX],1           ; 表示状態
                MOV     SI,GEDSPSW
                CALL    FCALL
```

```
                MOV     AH,01H              ; 確認のため KB 入力
                                            ; 待ち
                INT     21H
                MOV     SI,GTERM            ; グラフィックス
                                            ; の終了
                CALL    FCALL
SMPL_END:       MOV     AX,4C00H
                INT     21H
;           グラフィックスドライバの存在を確認
;           (キャリーフラグが 1 ならドライバは存在しない)
DRV_CHK         PROC    NEAR
                MOV     AX,3D00H            ; ドライバをオープン
                MOV     DX,OFFSET DS:GDDEV_NAME
                INT     21H
                JC      DRV_CHK_END         ; オープンに失敗:
                                            ; 存在しない
                                            ; オープンに成功:
                                            ; 存在する
                MOV     BX,AX               ; ドライバをクローズ
                MOV     AH,3EH
                INT     21H
                CLC
DRV_CHK_END:    RET
DRV_CHK         ENDP
;           グラフィックスドライバのエントリテーブルのアドレスを取得する
;           (GRAPH_ADDR にエントリテーブルのセグメントアドレスを格納)
GETGD_ENT       PROC    NEAR
                MOV     AX,0
                MOV     BX,OFFSET DS:GRAPH_ADDR
                INT     0CDH
                RET
GETGD_ENT       ENDP
;           グラフィックスドライバの機能を呼び出す
;           (SI にエントリテーブル上のオフセットを設定して呼び出す)
FCALL           PROC    NEAR
                MOV     BX,OFFSET DS:PARA_AREA
                PUSH    DS                  ;DS 退避
                PUSH    DS                  ; グラフィックスドラ
                                            ; イバへのパラメータ
                PUSH    BX
                LDS     BS,DS:GRAPH_ADDR
                CALL    DWORD PTR DS:[BX+SI]
```

```
                POP     DS                      ;DS 復帰
                RET
FCALL           ENDP
CODE            ENDS
                END     START
```

■ C 言語での使用例

```
#include        <stido.h>
#include        <dos.h>
#include        <conio.h>

    union GrpDataTag  {
          unsigned int wk[1024];  /* データ領域の確保 */
          struct PsetTag{         /* 点描画のパラメータブロック */
                  unsigned long  Reserve;
                  unsigned int   Rop;
                  unsigned char  Act_mode;
                  unsigned int   X;
                  unsigned int   Y;
                  unsigned long  Color;
          }PsetTag;
          struct DspSw{           /* 表示スイッチのパラメータブロック */
                  unsigned long  Reserve;
                  unsigned char  Switch;
          }DspSw;
    }GrpDataTag;
    struct FuncEntryTag{          /* ファンクションの定義 */
          int(far pascal*FUNC[41])(unsigned long);
    }far*GL;
    FILE    *fp;
    union  REGS  inregs,outregs;
main(){
/* グラフィックスドライバの存在を確認する */
if      ((fp=fopen("GRAPH $¥0","r"))!=0){
         fclose(fp);                    /* 存在する */
         }else{
         return(0);                     /* 存在しない */
         };
/* エントリテーブルの先頭アドレスの取得 */
inregs.x.ax      =0;
inregs.x.bx      =(int)&GL;
int 86(0xcd,&inregs,&outregs);
```

```
                                              /* グラフィックスの開始 */
(*(GL->FUNC[0]))((unsigned long)(int far*)&GrpDataTag);
                                              /* 点の描画 */
GrpDataTag.PsetTag.Reserve      =0;           /* 予約パラメータ */
GrpDataTag.PsetTag.Rop          =0;           /* ラスタオペレーション番号 */
GrpDataTag.PsetTag.Act_mode     =1;           /* 動作番号 */
GrpDataTag.PsetTag.X            =400;         /* 点を描画する X 座標 */
GrpDataTag.PsetTag.Y            =100;         /* 点を描画する Y 座標 */
GrpDataTag.PsetTag.Color        =7;           /* パレット番号 */
(*(GL->FUNC[15]))((unsigned long)(int far*)&GrpDataTag);
GrpDataTag.DspSw.Switch         =1;           /* 表示スイッチの設定 */
(*(GL->FUNC[11]))((unsigned long)(int far*)&GrpDataTag);
getche();                                     /* 確認のための KB 入力待ち */
                                              /* グラフィックスの終了 */
(*(GL->FUNC[1]))((unsigned long)(int far*)&GrpDataTag);
};
```

第5章

EMSインターフェイス

5.1 イントロダクション

アプリケーションによっては、通常のメモリ容量の最大値（640Kバイト）よりも、大きなメモリ領域を必要とするものがあります。このため拡張メモリ（1Mバイト以上の空間）をアプリケーションで使用できるようにしたものが、拡張メモリ仕様（EMS）です。

EMSは、拡張メモリマネージャ（EMM）と呼ばれる拡張メモリを管理するデバイスドライバと、拡張メモリを使用するアプリケーションとのインターフェイスからなるソフトウェアです。

MS–DOSでは、このEMSを制御するソフトウェアを〝EMM.SYS〟および〝EMM386.SYS〟というデバイスドライバで提供しています。EMSの機能を利用するには、〝EMM.SYS〟または〝EMM386.SYS〟をCONFIG.SYSファイルに組み込みます。デバイスドライバの詳細な組み込み方法については「MS–DOSユーザーズリファレンスマニュアル」を参照してください。

ここでは拡張メモリの概念、プログラミングの注意事項、各ファンクションの機能の説明などをしていきます。また、本章の5.9「用語解説」ではこの章で使われている用語についてまとめてあります。必要に応じて参照してください。

■ 拡張メモリとは

拡張メモリは、MS–DOSの640Kバイト（ハイレゾリューションモードでは768Kバイト）の制限を超えたメモリです。EMSは論理上、32Mバイト（PC–9800シリーズでは最大14.5Mバイト）までの拡張メモリを利用することができます。V30、8086、8088、80286、386/386SX、486/486SX（リアルモード）のマイクロプロセッサ（CPU）は、物理的に1Mバイトのアドレスしか持つことができませんが、その物理アドレスの範囲にある〝ウィンドウ〟を通じて拡張メモリにアクセスすることができます。

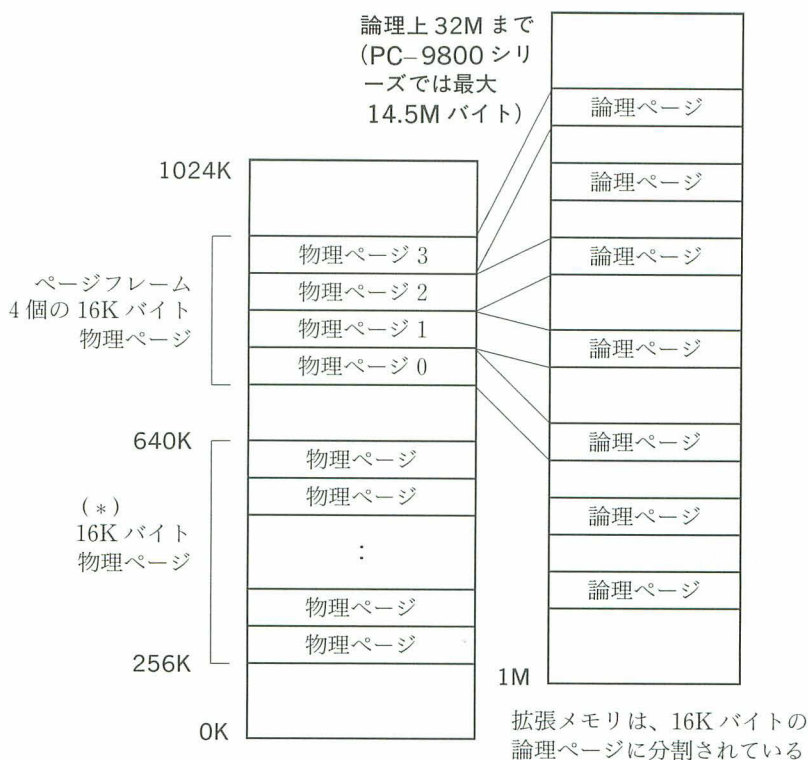
■ 拡張メモリの働き

拡張メモリは、〝論理ページ〟と呼ばれるメモリのブロックに分割されます。ひとつの論理ページの、典型的な大きさは16Kバイトです。コンピュータは、〝ページフレーム〟と呼ばれる物理的なメモリのブロックを通して論理ページをアクセスします。ページフレームは、多数の物理ページ、すなわちマイクロプロセッサが直接的にアドレスを付けることができるページを含みます。物理ページの典型的な大きさも、16Kバイトです。

このページフレームは、拡張メモリに対して〝ウィンドウ〟として機能します。

ちょうど、ディスプレイが大きなスプレッドシートのウィンドウであるように、ページフレームは、拡張メモリに対するウィンドウになっています。

拡張メモリの論理ページは、ページフレームのある物理ページにマッピング（実際に存在するかのように作る）されます。したがって、実際の物理ページに対する読み込みや書き出しは、関連する論理ページへの読み込みや書き出しになります。ひとつの論理ページは、ある物理ページに対するページフレームにマッピングされます。

次の図は、ページフレーム、物理ページ、論理ページの関係を示しています。なおこの図は概念的なものであり、PC–9800シリーズのEMSドライバでは640Kバイトよりも下位のアドレスにはページフレームがありません。

(＊) OS（オペレーティングシステム）に限って使用できることを意味している。

ページフレームは、640Kバイトよりも上位のアドレスに位置しています。通常640K〜1Mバイトの間には、VRAMや拡張ROM空間があります。

EMSでは、OSに対して640Kバイトよりも下位のアドレスにある物理ページを通じて、拡張メモリにアクセスする方法も定義しています（ただし、PC–9800シリーズのEMSドライバでは使用できません）。これらの方法は、OSを発展させるものとしてのみ意味を持ちます。

PC–9800シリーズではページフレームのアドレス（位置）とサイズは、機種やハードウェアモードによって次のように異なります。

ハードウェアモード	ページフレームアドレス	物理ページ数
ノーマルモード機種(*1)	B0000H～BFFFFH(*2)	4ページ
	C0000H～CFFFFH(*3)	
	C0000H～C7FFFH(*3)	2ページ
	C8000H～CFFFFH(*3)	
ハイレゾリューションモード	B0000H～BFFFFH	4ページ

(*1) ページフレームアドレスは、使用機種、使用するEMSドライバによって異なります。詳しくは「MS–DOSユーザーズリファレンスマニュアル」を参照してください。

(*2) B0000H～BFFFFHは、グラフィックVRAMと重なる（EMS用ページフレームとバンク切り換えによって使用している）ため、アプリケーション自身がページフレームとVRAMを切り換えて使用する必要があります。このVRAMとページフレームの切り換えは、EMSドライバを呼び出すことによって行うことができます。詳しくは、ファンクション7000H、7001Hの説明を参照してください。また、CPUがV30/8086の機種では使用できません。

(*3) CPUがV30/8086の機種では、EMS機能付き増設RAMボードが必要です。

5.2　拡張メモリを使用するプログラムの書き方

ここでは、拡張メモリを使用するために、まず最初にすべてのプログラムが実行しなければならないことについて説明します。

拡張メモリを使用するために、アプリケーションでは基本ファンクションの機能を使用して、次のような手続きを実行しなければなりません。

1. EMMがインストールされているかどうかを確かめる（ファンクション40H）。

2. 拡張メモリのページがアプリケーションに対して十分な容量が存在するかどうか判断する（ファンクション42H）。

3. 拡張メモリのページをアロケートする（ファンクション43Hまたは51H）。

4. 使用する各物理ページの基底アドレスを求める（ファンクション5800H）。

5. 拡張メモリを物理ページにマップする（ファンクション44Hまたは5000H）。

6. 物理ページを標準メモリと同じようにアクセスし、拡張メモリのデータ読み込み、書き出し、実行

を行う。

7. 拡張メモリの使用を終了する前に、拡張メモリのページをデアロケートする（ファンクション45Hまたは51H)。

ファンクションの呼び出し方法は、レジスタAHにコール番号とその他の必要な情報をセットして、INT 67Hを実行します。
また、各ファンクションの詳しい説明はファンクション一覧を参照してください。

5.3　プログラミング上の注意事項

ここでは、EMSインターフェイスを使用するアプリケーションを開発する際の、注意事項について説明します。

・拡張メモリをプログラムスタックに使用してはいけません。

・割り込み67Hを変更してはいけません。これはEMSインターフェイス使用時の割り込みベクタです。割り込み67Hを変更すると、EMSインターフェイスが使用できなくなります。

・アプリケーションで、標準メモリにある物理ページを使用してはいけません。
もし使用する場合は、アプリケーションが標準メモリにある物理ページのメモリ空間を確保しなければなりません。確保しないで使用すると、プログラムの暴走の原因となります。

・同じEMMハンドルの同じ論理ページを、複数の物理ページにマップする場合に注意が必要です。この場合、物理ページにデータを書き込むと、他の物理ページにマップされている同じ論理ページにも、自動的にデータが書き込まれる場合と、書き込まれない場合があります。どちらの状態であるかは、同じ論理ページを別々の物理ページにマップし、片方の物理ページにデータを書き込み、もう片方の物理ページにも書き込まれているかで確認します。
これらは、拡張メモリのデータエイリアシング（同じ論理ページを別々の領域として扱うこと）を行う場合に注意しなければなりません。

・アプリケーションは、終了時に必ずEMMハンドルをデアロケートしなければなりません。これらのデアロケートした論理ページは、他のアプリケーションで使用できるようになります。もし、デアロケートしないと、その論理ページはEMMハンドルにアロケートされたままになり、他のアプリケーションで使用できなくなります。
また、このようなことが繰り返し起こると、論理ページまたはEMMハンドルを使い果たすことになり、EMSインターフェイスが使用できなくなります。

・TSR（Terminate and Stay Resident Programs：終了後もメモリに残るプログラムで、常駐プログラムともいう）は、論理ページのマップを行う前に、必ず物理ページのマップ状態を保存するようにし

ます。TSR が拡張メモリを使用している可能性のある他のプログラムに割り込みをかける場合があるので、最初に物理ページのマップ状態を保存しないで論理ページをマップしてはいけません。また、終了する前には TSR は物理ページのマップ状態を元に戻さなければなりません。

・アプリケーションで用意する保存領域は、論理ページをマップする物理ページ上に設けてはいけません。ただし、ファンクション 55H（ページマップの変更とジャンプ）とファンクション 56H（ページマップの変更とコール）は例外です。

5.4 応用ファンクションの機能

EMS は基本ファンクションに加えて、拡張メモリを使用するソフトウェアの能力を高めるいくつかの応用ファンクションを備えています。

ここでは、それらのプログラミングで利用するファンクションの機能について説明します。

注意　プログラムで応用ファンクションを使用する前に、まずインストールされている EMM が、これらのファンクションをサポートしているバージョンであるかどうかをファンクション 46H（バージョンの取得）を使用して調べてください。

■ 物理ページのマッピングの状態を保存する

割り込みサービスルーチン、デバイスドライバ、常駐ソフトウェアのようなソフトウェアは、次のような処理を行わなければなりません。

・物理ページの現在のマッピング状態の保存
・マッピングコンテキストの切り換え
・拡張メモリの区域の操作
・物理ページのマッピング状態の復元

物理ページの状態を保存するには、ファンクションの 47H と 48H、またはファンクションの 4E00H、4E01H と 4F00H、4F01H を使用します。

次に、物理ページのマッピング状態を保存し、復元する 3 つの方法について説明します。

1. ページマップのセーブ（ファンクション 47H）とページマップのリストア（ファンクション 48H）
   これは 3 つの方法での中でいちばん簡単な方法です。ファンクション 47H で物理ページの状態を保存し、ファンクション 48H で物理ページの状態を復元します。物理ページの状態を保存するメモリ領域は EMM の内部にあるので、アプリケーション側で用意する必要はありません。
   物理ページの状態を復元するとき、ファンクション 47H で使用した EMM ハンドルをファンクション 48H で指定しないと、正しく復元されませんので注

意してください。

この方法は、一度保存すると復元まで同じEMMハンドルを使用して保存することができません。復元を行うと、再度同じEMMハンドルに保存することができます。

また、保存される物理ページの状態はEMS標準（64Kバイト）のページフレーム内にある物理ページのマッピング状態のみです。

2. ページマップの取得／設定（ファンクション4E00H、4E01H）

この方法はアプリケーションが用意したメモリ領域に物理ページのマッピング状態を保存します。この方法で物理ページ状態を復元するには、アプリケーションが以前に物理ページ状態を保存した、メモリ領域のアドレスを指定します。

もし、アプリケーションが復元する前に、さらに物理ページのマッピング状態を保存する必要がある場合は、この方法を使用します。

3. システム内の指定のマップ可能メモリ領域用のマッピングコンテキストの一部をセーブ／リストアする（ファンクション4F00H、4F01H）

この方法は指定した物理ページのマッピング状態を保存する方法です。アプリケーションがすべての物理ページのマッピング状態を保存する必要がないとき、この方法を使用します。このファンクションでは、保存するメモリ領域をアプリケーション側で用意してください。

■ハンドルの検索とページ数

ある種のユーティリティプログラムでは、どのような方法で拡張メモリが使用されつつあるかを記録した軌跡を保持する必要があります。これを実行するにはファンクション4BH、4CH、4DHを使用します。

■複数ページのマッピングとアンマッピング

複数ページのマッピングは、あるアプリケーションのマッピング中にかかるオーバーヘッドを減少させます。複数ページのマッピングとアンマッピングにはファンクション5000H、5001Hを使用します。

さらに、物理ページの代わりに、セグメントアドレスを使用してマッピングすることもできます。たとえば、ページフレーム基底アドレスがC000Hにセットされていれば、物理ページ0かセグメントC000Hへマッピングすることができます。すべての拡張メモリの物理ページと、それらに対応するセグメント値のクロスリファレンスを得るには、ファンクション5800H、5801Hを使用します。

■ページのリアロケート

ファンクション51Hを使用すると、EMMハンドルに割り当てられている論理ページ数を変更することができます。

■ ハンドルの使用とハンドルへの名前の割り当て

EMM ハンドルには 8 文字（8 バイト）の名前であるハンドル名を付けることができます。ハンドル名を付けることにより、複数のアプリケーションで使用する共通の拡張メモリを設けることができます。

たとえば、複数のアプリケーションで共通に使用する EMM ハンドルにハンドル名を付けて、共通の EMM ハンドルにします。EMM は共通の EMM ハンドルをハンドル名で捜し、捜し出した EMM ハンドルの論理ページをアクセスすることで共通の拡張メモリが設けられます。

ハンドル名を EMM ハンドルに付けるには、ファンクション 5300H、5301H を使用します。また、特定のハンドル名が付けられている EMM ハンドルを得るには、ファンクション 5400H～5402H を使用します。さらに、EMM ハンドルに割り当てられた論理ページ数を調べるにはファンクション 4DH を使用します。

■ ハンドルの属性の使用

EMM ハンドルには名前をつけるだけでなく、属性（揮発性／不揮発性）を付けることができます。これはファンクション 5201H を使用します。

属性が不揮発性とは、拡張メモリのデータはウォームブート後でも保持することができる属性のことです。属性が揮発性ではウォームブートの前のデータはウォームブート後は保持されません。ハンドルの初期属性は揮発性になっています。

このファンクションを利用する場合は、そのシステムに組み込まれた拡張メモリのハードウェアに依存しています。したがって、ハンドルのページに属性を割り当てる前にファンクション 5202H を使用し、ハードウェアの属性を調べる必要があります。

■ ページマップの変更とジャンプ／コール

拡張メモリに格納されているプログラムを実行するには、ファンクション 55H または、56H を使用します。

このファンクションはプログラムが格納されている拡張メモリの論理ページをマッピングし、ジャンプまたはコールを行います。

拡張メモリにプログラムを格納することにより、標準メモリの容量を超えるプログラムを実行することや使用する標準メモリの容量を少なくすることができます。

■ メモリ領域の移動と交換

ファンクション 5700H、5701H を使用すると、標準メモリと拡張メモリの間で簡単にデータ移動／交換することができます。このファンクションは 1 回の呼び出しで、1M バイトまでのデータを扱うことができます。

アプリケーションはこのファンクションがなくても、同じ処理を実行することができますが、拡張メモリマネージャを使用するとアプリケーションのオーバーヘッドが減少します。

■ 物理ページの数と各物理ページのアドレスを得る

ファンクション 5800H、5801H を使用すると、物理ページの数と各物理ページ

のアドレスを調べることができます。またサブファンクションとして物理ページの数を返すものと、各物理ページのアドレスを返すものがあります。

■ OSファンクション

アプリケーションに対するファンクションに加えて、EMSではOSに対するファンクションも定義しています。これらのファンクションはOSによっては無効にされる場合もあります。したがって、プログラムがこれらのファンクションに過度に依存することは、望ましいことではありません。この警告を無視して、このファンクションを使用するアプリケーション（OSも含む）は、他のプログラムと互換性のないものになる危険が高くなります。

5.5 EMSファンクション一覧

EMSインターフェイスには次のような機能が用意されています。

ファンクション		機　能
基本	40H	ステータスの取得
	41H	ページフレームのアドレスの取得
	42H	未アロケートページ数の取得
	43H	ページのアロケート
	44H	ハンドルページのマップ／アンマップ
	45H	ページのデアロケート（開放）
	46H	バージョンの取得
拡張	47H	ページマップのセーブ
	48H	ページマップのリストア
	49H、4AH	システム予約
	4BH	ハンドル数の取得
	4CH	ハンドルページの取得
	4DH	全ハンドルページの取得
	4E00H	ページマップの取得
	4E01H	ページマップの設定
	4E02H	ページマップの取得と設定
	4E03H	ページマップセーブ配列のサイズ取得
	4F00H	ページマップの一部をセーブ
	4F01H	ページマップの一部をリストア
	4F02H	ページマップの一部をセーブする配列のサイズ取得
	5000H	複数ハンドルページのマップ／アンマップ（論理ページ／物理ページ方式）
	5001H	複数ハンドルページのマップ／アンマップ（論理ページ／セグメントアドレス方式）
	51H	ページの再アロケート

ファンクション		機　能
拡張	5200H	ハンドルアトリビュートの取得
	5201H	ハンドルアトリビュートの設定
	5202H	ハンドルアトリビュートのケイパビリティの取得
	5300H	ハンドル名の取得
	5301H	ハンドル名の設定
	5400H	ハンドルのディレクトリ取得
	5401H	指定ハンドルのサーチ
	5402H	ハンドルの総数の取得
	55H	ページマップの変更とジャンプ
	56H	ページマップの変更とコール
	5602H	ページマップスタックサイズの取得
	5700H	メモリ領域の移動
	5701H	メモリ領域の交換
	5800H	マップ可能な物理アドレス配列の取得
	5801H	マップ可能な物理アドレス配列エントリの取得
	*5900H	ハードウェア構成配列の取得
	*5901H	未アロケートのローページ数の取得
	5A00H	標準サイズのページのアロケートと固有の EMM ハンドルの割り当て
	5A01H	ローページのアロケートと固有の EMM ハンドルの割り当て
	*5B00H	代替マップレジスタセットの取得
	*5B01H	代替マップレジスタセットの設定
	*5B02H	代替マップセーブ配列のサイズ取得
	*5B03H	代替マップレジスタセットのアロケート
	*5B04H	代替マップレジスタセットの開放
	*5B05H	DMA レジスタセットのアロケート
	*5B06H	代替マップレジスタ上の DMA の使用許可
	*5B07H	代替マップレジスタ上の DMA の使用不許可
	*5B08H	DMA レジスタセットの開放
	5CH	ウォームブートのための拡張メモリの準備
	*5D00H	OS/E ファンクションセットの使用許可
	*5D01H	OS/E ファンクションセットの使用不許可
	*5D02H	アクセスキーのリターン
	7000H	ページフレーム用バンクのステータスの取得
	7001H	ページフレーム用バンクの状態の設定

なお、ファンクションについている（*）は OS のみが使用可能なものです。
これ以降では、各ファンクションごとにその使用方法を解説します。

INT 67H

ファンクション
40H　ステータスの取得

コール　AH = 40H

リターン

AH = 00H	**正常実行**	（メモリマネージャあり、ハードウェアは正常に動作）
= 80H	**回復不可**	（メモリマネージャソフトウェアが動作不能）
= 81H	**回復不可**	（拡張メモリハードウェアが動作不能）
= 84H	**回復不可**	（メモリマネージャに渡されたファンクションが未定義）

解　説

メモリマネージャが存在し、ハードウェアが正しく動作しているかどうかを示すステータスコード値を返します。

サンプル

```
MOV   AH,40H               ;load function code
INT   67H                  ;call the memory manager
OR    AH,AH                ;check EMM status
JNZ   EMM_ERR_HANDLER      ;jump to error handler on error
```

INT 67H

ファンクション
41H ページフレームのアドレスの取得

コール AH = 41H

リターン
AH = 00H 正常実行（ページフレームアドレスが BX に返された）
= 80H 回復不可（メモリマネージャソフトウェアが動作不能）
= 81H 回復不可（拡張メモリハードウェアが動作不能）
= 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）
BX =ページフレームセグメントアドレス（AH に 0 が返ったときのみ）

解　　説

ページフレームが位置しているセグメントアドレスを返します。

サンプル

```
page_frame_segment              DW ?

MOV    AH,41H                   ;load function code
INT    67H                      ;call the memory manager
OR     AH,AH                    ;check EMM status
JNZ    EMM_ERR_HANDLER          ;jump to error handler on error
MOV    page_frame_segment,BX    ;save page frame address
```

INT 67H

ファンクション
42H 未アロケートページ数の取得

コール

AH = 42H

リターン

AH = 00H　**正常実行**（未アロケートページ数と総ページ数がBXとDXに返された）
　　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
BX =未アロケートページ数
DX =総ページ数

解　説

プログラムが使用できるアロケートされていない論理ページ数と、拡張メモリの総論理ページ数の値を返します。

サンプル

```
un_alloc_pages          DW ?
total_pages             DW ?

MOV   AH,42H                     ;load function code
INT   67H                        ;call the memory manager
OR    AH,AH                      ;check EMM status
JNZ   EMM_ERR_HANDLER            ;jump to error handler on error
MOV   un_alloc_pages,BX          ;save unallocated page count
MOV   total_pages,DX             ;save total page count
```

INT 67H

ファンクション 43H ページのアロケート

コール

AH ＝ 43H
BX ＝アロケートしたいページ数

リターン

AH ＝ 00H　**正常実行**（ページがアロケートされた）
　＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　＝ 85H　**復旧可能**（全 EMM ハンドルは使用中）
　＝ 87H　**復旧可能**（プログラムの要求を満たすような拡張メモリページがシステム内に存在しない）
　＝ 88H　**復旧可能**（プログラムの要求を満たすような未アロケートページがない）
　＝ 89H　**復旧可能**（プログラムが 0 ページをアロケートしようとした）
DX ＝ EMM ハンドル
　以降このハンドルを使用します（ハンドルは 255 まで）。ハンドルの上位バイトは 0 で、アプリケーションでは使用できません。

解　説

要求されたページ数をアロケートし、アロケートしたページに EMM ハンドルを割り付けます。その EMM ハンドルはアプリケーションがページを開放するまで、そのページを保有します。

このファンクションを使用して割り付けされたハンドルは、16K バイトのページを所有します。このサイズは拡張メモリの標準サイズです。拡張メモリボードハードウェアが 16K バイトサイズのページの供給ができない場合は、標準サイズではないページをいくつか結合して 16K バイトページの形になるようにします。

EMM が返すハンドルの数値は、10 進の 1～254（0001H～00FEH）の範囲です。ただしハンドル 0（OS ハンドル）が、このファンクションで返されることはありません。また、ハンドルの最上位バイトの 00H は、アプリケーションで使用することはできません。メモリマネージャは 255（ハンドル 0 を含む）までのハンドルを供給でき、EMM がサポートしているハンドルは、ファンクション 5402H（ハンドルの総数の取得）を使用して知ることができます。

このファンクションではハンドルに 0 ページをアロケートすることはできません。アプリケーションが 0 ページをアロケートする場合は、ファンクション 5A00H（標準サイズのページのアロケート）を使用します。

注意　次の事項は、拡張メモリマネージャインプリメンタとOS開発者のみに関係するものです。アプリケーションのユーザーは、このメモリマネージャの特質を利用することはできません。これらの規則を守らない場合は、マイクロソフト社のOSと互換性がなくなります。

拡張メモリマネージャにはこの仕様に互換性をもたせるために、OSのみが使用可能なハンドルが準備されています。このハンドルの値は0000Hで拡張メモリマネージャがインストールされたときに、このハンドルにアロケートされたページの1セットが与えられます。メモリマネージャが0000Hのハンドルに自動的にアロケートするページは、メモリの上位アドレスに存在し、一般的にこれは40000H（256K）から9FFFFH（640K）のアドレス間にあります。しかし、この範囲でハードウェアやメモリマネージャをサポートしている場合は、この範囲を上下に拡張することができます。拡張メモリデバイスドライバがインストールされると、このハンドルはすでに存在していると想定され、すぐに使用可能になります。したがって、このハンドルを得るために本ファンクションを呼び出す必要はありません。
OSがこのハンドルを使用する場合は、0000Hの特別なハンドル値を使用して任意のEMMファンクションを呼び出すことができます、また、このハンドルにページをアロケートするためにはファンクション51H（ページの再アロケート）を呼び出します。
このハンドルには、次のように2つの特別な場合があります。

1. ファンクション43H（ページのアロケート）
   このファンクションはハンドル値として0を返すことはありません。アプリケーションにおいてはページをアロケートし、そのページをもつハンドルを得るためにファンクション43Hを呼び出さなければなりません。ファンクション43Hは0のハンドル値を返すことはないので、アプリケーションでこの特別なハンドルにアクセスすることはできません。

2. ファンクション45H（ページのデアロケート（開放））
   OSが特別なEMMハンドルにアロケートされたページの解除のために、このファンクションを使用するとEMMハンドルが所有するページは使用可能になり、メモリマネージャに返されます。しかし、このEMMハンドルを再び割り当てることはできません。メモリマネージャはこのファンクションのEMMハンドルへの要求を、ファンクション43H（ページのアロケート）の要求と同じものとして取り扱います。したがって、このEMMハンドルへのリアロケートのページ番号は0になります。

サンプル

```
num_of_pages_to_alloc            DW ?
emm_handle                       DW ?

MOV   BX,num_of_pages_to_alloc   ;load number of pages needed
MOV   AH,43H                     ;load function code
INT   67H                        ;call the memory manager
OR    AH,AH                      ;check EMM status
JNZ   EMM_ERR_HANDLER            ;jump to error handler on error
MOV   emm_handle,DX              ;save EMM handle
```

INT 67H

ファンクション 44H ハンドルページのマップ／アンマップ

コール

AH ＝ 44H

AL ＝**物理ページ番号**

論理ページをマップする物理ページの番号。

物理ページは相対的に0から番号付けされています。

BX ＝**論理ページ番号**

ページフレーム内の物理ページにマッピングされる論理ページの番号。論理ページは相対的に0から番号付けされています。論理ページは0から（EMMハンドルにアロケートされたページ数–1）範囲をとります。ただしBXの論理ページ番号にFFFFHをセットすると、AL内に定義されている物理ページを介して、論理ページへアクセスすることができなくなります（アンマップ）。

DX ＝ **EMMハンドル**（ファンクション43Hで取得したハンドル）

リターン

AH ＝ 00H **正常実行**（ページはマッピングされた）

＝ 80H **回復不可**（メモリマネージャソフトウェアが動作不能）

＝ 81H **回復不可**（拡張メモリハードウェアが動作不能）

＝ 83H **回復不可**（指定のEMMハンドルがない）

＝ 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）

＝ 8AH **復旧可能**（指定の論理ページがEMMハンドルにアロケートされている論理ページの範囲を超えた、または論理ページがアロケートされていない）

＝ 8BH **復旧可能**（物理ページ番号が実際の物理ページの範囲を超えた。範囲内にある物理ページにマッピングし直すことにより復旧が可能）

解　説

システムメモリ内のマッピング可能な領域にある指定の物理ページに、論理ページをマッピングします。物理ページ番号の最小値は実メモリ範囲内のメモリ領域と関連しています。システム内のどの物理ページがマッピング可能か、また指定の物理番号に対応するセグメントアドレスを調べるためには、ファンクション5800H（マッピング可能な物理アドレス配列の取得）を使用します。このファンクションは物理ページ番号とセグメントアドレス間のクロスリファレンスを準備します。

また、論理ページにFFFFHをセットすることにより、物理ページをアンマッピングすることも可能です（物理ページの解除）。アンマッピングした場合は、物理ページを介して論理ページへアクセス（読

み書き）することができないので、物理ページへのアクセス（読み書き）は行わないでください。

たとえばプログラムを読み込んだり実行したりする前に、マッピングされたページすべてを解除します。そうすることにより、読み込んだプログラムが拡張メモリにアクセスしても、その前のプログラムでマッピングしたページにアクセスすることがありません。しかし物理ページを解除する前にマッピングコンテキストを保存すると、後で復元を行いアクセス（読み書き）することができます。マッピングコンテキストを保存するにはファンクション47H（ページマップのセーブ）、4E00H〜4E03H、4F00H〜4F02Hを用います。マッピングコンテキストを復元するにはファンクション48H（ページマップのリストア）、4E00H〜4E03H、4F00H〜4F02Hを用います。

サンプル

```
logical_page_number              DW ?
physical_page_number             DB ?
emm_handle                       DW ?

MOV    DX,emm_handle             ;load EMM handle
MOV    BX,logical_page_number    ;load logical page number
MOV    AL,physical_page_number   ;load physical page number
MOV    AH,44H                    ;load function code
INT    67H                       ;call the memory manager
OR     AH,AH                     ;check EMM status
JNZ    EMM_ERR_HANDLER           ;jump to error handler on error
```

INT 67H

ファンクション 45H ページのデアロケート（開放）

コール

AH ＝ 45H
DX ＝ EMMハンドル（ファンクション43Hで取得したハンドル）

リターン

AH ＝ 00H **正常実行**（指定のEMMハンドルは解除された）
＝ 80H **回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H **回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H **回復不可**（指定のEMMハンドルがない）
＝ 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 86H **復旧可能**（ページマッピングコンテキストのセーブまたはリストア（ファンクション47Hまたは48H）でエラーになった。指定のEMMハンドル用のセーブエリア内にページマップレジスタのステータスが含まれている。ファンクション47H（ページマップのセーブ）でセーブしたが、ファンクション48H（ページマップのリストア）を実行していなかった（マッピングコンテキストをセーブしたら、EMMハンドルを開放する前にリストアしなければならない））

解　説

現在EMMハンドルに割り付けられている論理ページを開放します。アプリケーションがあるページの開放を行うことにより、他のアプリケーションがそのページを使用できるようになります。また、ハンドルが開放された場合はハンドル名はすべてNULL（00H）にセットされます。

注意　アプリケーションは、MS-DOSへ返る前に必ずこのファンクションを実行しなければなりません。もし実行しなければ、これらのページやEMMハンドルを他のプログラムで使用することはできません。拡張メモリを使用するプログラムに致命的なエラーが発生したり、強制終了した場合にページがアロケートされている可能性がある場合は、トラップしなくてはなりません。

サンプル

```
emm_handle          DW ?

MOV   DX,emm_handle      ;load EMM handle
MOV   AH,45H             ;load function code
INT   67H                ;call the memory manager
OR    AH,AH              ;check EMM status
JNZ   EMM_ERR_HANDLER    ;jump to error handler on error
```

INT 67H

ファンクション 46H バージョンの取得

コール　AH = 46H

リターン　AH = 00H　正常実行（バージョン番号がALに返された）
　= 80H　回復不可（メモリマネージャソフトウェアが動作不能）
　= 81H　回復不可（拡張メモリハードウェアが動作不能）
　= 84H　回復不可（メモリマネージャに渡されたファンクションが未定義）
AL =バージョン番号

解説

EMSインターフェイスのバージョン番号を取得します。

バージョン番号はBCD形式で返されます。上位4ビットがバージョン番号の整数部分を表し、下位4ビットが小数点以下の部分を表します。

たとえばバージョン4.0の場合は、ALは40Hになります。

```
      AL
ビット 76543210
      01000000
      └─┬┘└─┬┘
       4  .  0
```

サンプル

```
emm_version                     DW ?

MOV   AH,46H                    ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
MOV   emm_version,AL            ;save emm version
```

INT 67H

ファンクション 47H ページマップのセーブ

コール

AH ＝ 47H
DX ＝ EMMハンドル
　ソフトウェアまたはハードウェア割り込みを、サービスする割り込みサービスルーチンに割り当てられているEMMハンドル。ただし割り込みサービスルーチンは、ページをマッピングする前にページマップハードウェアのステータスを保存しなければなりません。

リターン

AH ＝ 00H　**正常実行**（ページマップはセーブされた）
＝ 80H　**回復不可**（メモリマネージャやソフトウェアが動作不能）
＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H　**回復不可**（指定のEMMハンドルがない）
＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8CH　**復旧可能**（ページマップレジスタのステータスをストアするセーブエリア内に空がないため、マップレジスタのステータスはセーブされなかった）
＝ 8DH　**状況により復旧可能**（セーブエリアには、プログラムが指定したEMMハンドル用のセーブマップレジスタステータスがすでにセットされている）

解　説

ページマップレジスタ（ページのマッピング状態）の内容を内部エリアに保存します。

通常、このファンクションはソフトウェアまたはハードウェア割り込みが発生したときに、アクティブ状態のEMMハンドルのメモリマッピングコンテキストを保存するために使用されます。

拡張メモリを使用している常駐プログラムや割り込み処理ルーチン、デバイスドライバを書く場合、ハードウェアのマッピング状態を保存しなければなりません。なぜなら、そのプログラムがハードウェア割り込み、ソフトウェア割り込み、MS–DOSによって呼び出されたときに、拡張メモリを使用している他のアプリケーションソフトウェアが稼働している可能性があるためです。

また、常駐プログラムや割り込み処理ルーチン、デバイスドライバが初期化された場合は、割り当てられたEMMハンドルを必要とします。ただし、これは割り込まれた側のアプリケーションのEMMハンドルではありません。

保存されるページマップレジスタはページフレームのうち、EMSのバージョン3.xで定義されている64Kバイトのページフレームのみに関するマップレジスタの状態を保存します。EMSバージョン3.xの

仕様で書かれたすべてのアプリケーションは、この 64K バイトのものだけのマップレジスタの状態を保存するため、マッピング可能なページのすべてのマッピングコンテキストを保存するとメモリ効率が落ちます。64K バイトを超えるアプリケーションのページマツプレジスタの保存、復元はファンクション 4E00H〜4E03H または 4F00H〜4F02H を用います。

サンプル

```
emm_handle                              DW ?

MOV   DX,emm_handle                     ;load EMM handle
MOV   AH,47H                            ;load function code
INT   67H                               ;call the memory manager
OR    AH,AH                             ;check EMM status
JNZ   EMM_ERR_HANDLER                   ;jump to error handler on error
```

INT 67H

ファンクション 48H ページマップのリストア

コール

AH ＝ 48H
DX ＝ EMMハンドル
ソフトウェアまたはハードウェア割り込みサービスする割り込みサービスルーチンに割り当てられているEMMハンドル。割り込みサービスルーチンは、ページマッピッグの状態を復元しなければなりません。

リターン

AH ＝ 00H　**正常実行**（ページマップはリストアされた）
＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H　**回復不可**（指定のEMMハンドルがない）
＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8EH　**状況により復旧可能**（指定のEMMハンドル用のセーブエリア内にページマップレジスタがない。プログラムがページマッピングの状態をセーブしていないため、その内容をリストアできない）

解　説

固有なEMMハンドルのために、内部のセーブエリアからページマップレジスタの内容を復元します。

拡張メモリを使用している常駐プログラムや割り込み処理ルーチン、デバイスドライバを書くときには、ハードウェアのマッピング状態を保存しなければなりません。なぜなら、そのプログラムがハードウェア割り込み、ソフトウェア割り込み、MS-DOSによって呼び出されたときに、拡張メモリを使用している他のアプリケーションソフトウェアが稼働している可能性があるためです。

また、常駐プログラムや割り込み処理ルーチン、デバイスドライバが初期化された場合は、割り当てられたEMMハンドルを必要とします。ただし、これは割り込まれた側のアプリケーションのEMMハンドルではありません。

保存されるページマップレジスタはページフレームのうち、EMSのバージョン3.xで定義されている、64Kバイトのページフレームのみに関するマップレジスタの状態を保存します。EMSバージョン3.xの仕様で書かれたすべてのアプリケーションは、この64Kバイトのものだけのマップレジスタの状態を保存するため、マッピング可能なページのすべてのマッピングコンテキストを保存するとメモリ効率が落ちます。64Kバイトを超えるアプリケーションのページマップレジスタの保存、復元はファンクション4E00H～4E03Hまたは4F00H～4F02Hを用います。

サンプル

```
emm_handle                          DW ?

MOV   DX,emm_handle                 ;load EMM handle
MOV   AH,48H                        ;load function code
INT   67H                           ;call the memory manager
OR    AH,AH                         ;check EMM status
JNZ   EMM_ERR_HANDLER               ;jump to error handler on error
```

INT 67H

ファンクション 49,4AH　システム予約

EMSの初期バージョンでは、ファンクション49H、4AHはページマップレジスタのI/O配列を返します。現在はこの番号は予約されているので、新しいプログラムでは使わないでください。

このファンクションを用いているプログラムは、それらをサポートしているハードウェア上では正常に動きます。しかし、EMSのバージョン4.0でファンクション4F00Hから5D02Hを用いているプログラムでは、ファンクション49Hと4AHを使用してはいけません。またプログラムで新しいファンクション（ファンクション4F00Hから5D02H）とファンクション49H、4AHを一緒に使用すると、これらのファンクションは正常に動きません。

INT 67H

ファンクション 4BH　ハンドル数の取得

コール　AH = 4BH

リターン
AH = 00H　正常実行（ハンドル数は BX に返された）
　= 80H　回復不可（メモリマネージャソフトウェアが動作不能）
　= 81H　回復不可（拡張メモリハードウェアが動作不能）
　= 84H　回復不可（メモリマネージャに渡されたファンクションが未定義）
BX =オープンしている EMM ハンドルの数（OS のハンドル 0 を含む）
　この数は 255 を超えることはありません。

解　説

システム内のオープンしている EMM ハンドル（OS のハンドル 0 を含む）の数を返します。

サンプル

```
total_open_emm_handle          DW ?

MOV    AH,4BH                      ;load function code
INT    67H                         ;call the memory manager
OR     AH,AH                       ;check EMM status
JNZ    EMM_ERR_HANDLER             ;jump to error handler on error
MOV    total_open_emm_handles,BX   ;save total active handle count
```

INT 67H

ファンクション
4CH　ハンドルページの取得

コール

AH ＝4CH
DX ＝EMMハンドル

リターン

AH ＝00H　**正常実行**（指定ハンドルのページ数がBXに返された）
　＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝83H　**回復不可**（指定のEMMハンドルがない）
　＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
BX ＝**指定のEMMハンドルにアロケートされている論理ページ数**
　この数は2048を超えることはありません。

解　説

指定のEMMハンドルにアロケートされているページ数を返します。

サンプル

```
emm_handle                      DW ?
page_alloc_to_handle            DW ?

MOV   DX,emm_handle             ;load EMM handle
MOV   AH,4CH                    ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
MOV   page_alloc_to_handle,BX   ;save handle pages
```

INT 67H

ファンクション

4DH 全ハンドルページの取得

コール

AH　　= 4DH
ES:DI =オープンされているすべての EMM ハンドルのコピーと、各ハンドルにアロケートされているページ数がリストアされる配列のポインタ

```
handle_page_struct          STRUC
  emm_handle                DW ?
  pages_alloc_to_handle     DW ?
handle_page_struct          ENDS
```

リターン

AH = 00H　**正常実行**（全ハンドルページの情報が返された）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
BX =**オープンされている EMM ハンドルの総数**（OS のハンドル 0 を含む）
OS 用のハンドル 0 は、常にオープン状態のため値に 0 が返されることはありません。また 255 を超えることはありません。

解　説

オープンされている EMM ハンドルと、各ハンドルにアロケートされているページ数の配列を返します。

また、各ストラクチャのメンバは次のようになります。

.emm_handle

最初のメンバはオープンしている EMM ハンドルの値を含むワードです。このファンクションが返すハンドル値は 10 進で 0 から 255（0000H から 00FFH）の間で、ハンドルの上位バイトは常に 0 です。

.pages_alloc_to_handle

2 番目のメンバはオープンしている EMM ハンドルにアロケートされたページ数を含むワードです。

4CH/4DH

サンプル

```
handle_page                     handle_page_struct 255 DUP(?)
total_open_handles              DW ?

MOV    AX,SEG handle_page       ;set handle page segment
MOV    ES,AX                    ;
LEA    DI,handle_page           ;ES：DI handle page pointer
MOV    AH,4DH                   ;load function code
INT    67H                      ;call the memoty manager
OR     AH,AH                    ;check EMM status
JNZ    EMM_ERR_HANDLER          ;jump to error handler on error
MOV    total_open_handles,BX    ;save all handle page
```

INT 67H

ファンクション 4E00H ページマップの取得

コール

AX = 4E00H

ES：DI = dest_page_map

セグメント：オフセットの形式で示されるデスティネーション配列へのポインタを示します。要求された配列のサイズを決定するためには、ファンクション4E03Hを使用します。

リターン

AH = 00H　**正常実行**（ページマップを取得できた）

= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）

= 81H　**回復不可**（拡張メモリハードウェアが動作不能）

= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）

= 8FH　**回復不可**（ファンクションパラメータが無効）

dest_page_map

この配列はマッピングレジスタの状態を含みます。また、プログラムがファンクション4E01Hをコールしたときに、取得した状態に戻すために必要な追加情報も含まれています。

解　説

すべてのマッピング可能なメモリ領域（実メモリと拡張メモリ）のために、マッピングコンテキストをセーブします。これはデスティネーションの配列へ、拡張メモリのマッピングレジスタの内容をコピーすると実行します。なお、このファンクションはEMMハンドルを必要としません。

サンプル

```
dest_page_map                    DB ? DUP(?)

MOV   AX,SEG dest_page_map       ;set dest_page_map segment
MOV   ES,AX                      ;
LEA   DI,dest_page_map           ;ES：DI dest_page_map
MOV   AX,4E00H                   ;load function code
INT   67H                        ;call the memory manager
OR    AH,AH                      ;check EMM status
JNZ   EMM_ERR_HANDLER            ;jump to error handler on error
```

INT 67H

ファンクション 4E01H ページマップの設定

コール

AX　　= 4E01H
DS : SI = source_page_map
セグメント：オフセットの形式で示されるソース配列アドレスのポインタを示します。アプリケーションではマッピング状態を含む配列ポインタになります。

リターン

AH = 00H　**正常実行**（ページマップを設定した）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（ファンクションパラメータが無効）
= A3H　**回復不可**（ソース配列の内容が不正または渡されたポインタが無効）

解　　説

システム内の各拡張メモリのマップレジスタにソース配列の内容をコピーすることによって、すべてのマッピング可能なメモリ領域（実メモリと拡張メモリ）のためにマッピングコンテキストを復元します。マッピングコンテキストを保存または復元する必要があれば、このファンクションをファンクション 47H と 48H の代用として使用することができます。このときに EMM ハンドルを使用する必要はありません。

サンプル

```
source_page_map                    DB ? DUP(?)

MOV   AX,SEG source_page_map       ;set source page map segment
MOV   DS,AX                        ;
LEA   SI,dest_page_map             ;DS：SI source page map
MOV   AX,4E01H                     ;load function code
INT   67H                          ;call the memory manager
OR    AH,AH                        ;check EMM status
JNZ   EMM_ERR_HANDLER              ;jump to error handler on error
```

INT 67H

ページマップの取得と設定

コール

AX ＝4E02H
ES：DI ＝dest_page_map
　セグメント：オフセットの形式で示されるデスティネーション配列アドレスへのポインタ。現在のマップレジスタの内容はこの配列中にセーブされます。
DS：SI ＝source_page_map
　セグメント：オフセットの形式で示されるソースアドレス配列へのポインタ。この配列の内容はマップレジスタ内にコピーされます。

リターン

AH ＝00H　**正常実行**（ページマップの取得と設定をした）
　＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　＝8FH　**回復不可**（ファンクションパラメータが無効）
　＝A3H　**回復不可**（ソース配列の内容が不正または渡されたポインタが無効）

dest_page_map
　この配列はマッピングレジスタの状態を含みます。また、プログラムがファンクション 4E01H をコールしたときに、取得した状態に戻すために必要な追加情報も含まれています。

解　説

ページマップの取得とセットを行います。

このファンクションは、システム内の各拡張メモリのマップレジスタの内容をデスティネーション配列にコピーします。次にソース配列の内容を各拡張メモリのマップレジスタにコピーします。これによりカレントのマッピングコンテキストを保存し、すべてのマッピング可能なメモリ領域（内部、拡張メモリ共）のために、前のマッピングコンテキストを復元します。

サンプル

```
dest_page_map                           DB ?  DUP(?)
source_page_map                         DB ?  DUP(?)

MOV    AX,SEG dest_page_map             ;set dest page map segment
MOV    ES,AX                            ;
MOV    AX,SEG source_page_map           ;set source page map segment
MOV    DS,AX                            ;
LEA    DI,dest_page_map                 ;ES：DI dest page map
LEA    SI,dest_page_map                 ;DS：SI source page map
MOV    AX,4E02H                         ;load function code
INT    67H                              ;call the memory manager
OR     AH,AH                            ;check EMM status
JNZ    EMM_ERR_HANDLER                  ;jump to error handler on error
```

INT 67H

ページマップセーブ配列のサイズ取得

コール　AX = 4E03H

リターン　AH = 00H　**正常実行**（配列のサイズを取得した）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（ファンクションパラメータが無効）

AL = size_of_array
プログラムがファンクション 4E00H、4E01H、4E02H をコールするときに必要とされるメモリエリアのバイト数。このメモリエリアはアプリケーションで供給されます。

解　説

ファンクション 4E00H、4E01H、4E02H によって渡された配列に必要なメモリの大きさを返します。ページマップのセーブ配列のサイズは、拡張メモリシステムの構成やマネージャの動作に依存しています。そのため、サイズはメモリマネージャがロードされた後に取得されなければなりません。

サンプル

```
size_of_array                    DB ?

MOV    AX,4E03H                  ;load function code
INT    67H                       ;call the memory manager
OR     AH,AH                     ;check EMM status
JNZ    EMM_ERR_HANDLER           ;jump to error handler on error
MOV    size_of_array,AL          ;save size of array
```

INT 67H

ファンクション 4F00H ページマップの一部をセーブ

コール

AX　　= 4F00H

DS：SI = partial_page_map

ページマップの一部を示すストラクチャのポインタ。

```
partial_page_map_struct          STRUC
  mappable_segment_count         DW ?
  mappable_segment               DW (?) DUP (?)
partial_page_map_struct          ENDS
```

ES：DI = dest_array

要求された配列の大きさを決定するためには、ファンクション4F02Hを使用します。

リターン

AH = 00H　**正常実行**（ページマップを取得した）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8BH　**回復不可**（指定のセグメントの一部が、マップ不可能）
　= 8FH　**回復不可**（ファンクションパラメータが無効）
　= A3H　**回復不可**（ソース配列の内容が不正または渡されたポインタが無効）

dest_array

この配列はマッピングコンテキストの一部と、プログラムがファンクション4F01Hをコールしたときに、このコンテキストを元の状態へ復元するために必要な追加情報などが含まれます。

解説

システム内で指定されたマッピング可能なメモリ領域のマッピングコンテキストの一部を保存します。またマッピングコンテキストの一部のみを保存するため、ファンクション4E00Hに比べて、セーブエリア用に確保するメモリエリアも削減され処理速度も速くなります。このファンクションはこれらの処理を、デスティネーション配列に選択されたマッピングコンテキストの内容をコピーすることにより行います。

また、各ストラクチャのメンバは次のようになります。

.mappable_segment_count
このメンバはワードで、すぐあとに続くワード配列内のメンバ数を定義します。
この数はマッピングできるセグメントの数を超えてはなりません。

.mappable_segment
2番目のメンバは、ワードでセーブされるマッピングコンテキストを所有するマッピング可能なメモリ領域のセグメントアドレスを含みます。このセグメントアドレスは、マッピング可能なセグメントでなければなりません。どのセグメントがマッピング可能かを調べるには、ファンクション5800Hを使用します。

サンプル

```
partial_page_map              partial_page_map_struct<>
dest_array                    DB ? DUP(?)

MOV   AX,SEG partial_page_map ;partial page map segment
MOV   DS,AX                   ;
LEA   SI,partial_page_map     ;DS:SI partial page map point
MOV   AX,SEG dest_array       ;set dest array segment
MOV   ES,AX                   ;
LEA   DI,dest_array           ;ES:DI dest arrey point
MOV   AX,4F00H                ;load function code
INT   67H                     ;call the memory manager
OR    AH,AH                   ;check EMM status
JNZ   EMM_ERR_HANDLER         ;jump to error handler on error
```

INT 67H

ページマップの一部をリストア

コール

AX　　= 4F01H
DS：SI = source_array

セグメント：オフセットの形式を持つソース配列のポインタを示します。アプリケーションでは、マップレジスタステートの一部を含む配列をポイントしなければなりません。要求された配列のサイズを決定するためには、ファンクション 4F02H（ページマップの一部をセーブする配列のサイズ取得）を参照してください。

リターン

AH = 00H　**正常実行**（ページマップを設定した）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（ファンクションパラメータが無効）
= A3H　**回復不可**（ソース配列の内容が不正または渡されたポインタが無効）

解　　説

システム内で指定されたマッピング可能なメモリ領域のマッピングコンテキストの一部を復元します。またマッピングコンテキストの一部のみを復元するため、全システムのマッピングコンテキストを復元するファンクション 4E01H に比べて、メモリエリアも削減され処理速度も速くなります。このファンクションはソース配列の内容を選択されたマッピングコンテキストに、コピーすることによって設定を行います。

サンプル

```
MOV   AX,SEG source_array      ;source array segment
MOV   DS,AX                    ;
LEA   SI,source_array          ;DS：SI source array point
MOV   AX,4F01H                 ;load function code
INT   67H                      ;call the memory manager
OR    AH,AH                    ;check EMM status
JNZ   EMM_ERR_HANDLER          ;jump to error handler on error
```

INT 67H

ページマップの一部をセーブする配列のサイズ取得

コール

AX = 4F02H
BX =部分的にマップされるページ数
　この数は、ファンクション 4F00H（ページマップの一部をセーブする）の mappable_segment_count と同じになる。

リターン

AH = 00H　**正常実行**（配列のサイズを取得した）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8BH　**回復不可**（物理ページ数がシステム内の物理ページの範囲を超えた）
　= 8FH　**回復不可**（ファンクションパラメータが無効）
AL = size_of_partial_save_array
　プログラムがファンクション 4F00H、4F01H（ページマップの一部をセーブ／リストア）をコールするときに必要とされるバイト数。このエリアはアプリケーション側で用意しなければなりません。

解　説

ファンクション 4F00H、4F01H によって渡された配列に必要なメモリの大きさを返します。ページマップのセーブ配列のサイズは、拡張メモリシステムの構成やマネージャの動作に依存しています。したがってハードウェアの構成や動作間に開きがあるため、指定のメモリマネージャがロードされた後にサイズを調べなくてはなりません。

サンプル

```
number_of_page_to_map             DW ?
size_of_partial_save_array        DW ?

MOV   BX,number_of_page_to_map          ;set number of page to map
MOV   AX,4F02H                          ;load function code
INT   67H                               ;call the memory manager
OR    AH,AH                             ;check EMM status
JNZ   EMM_ERR_HANDLER                   ;jump to error handler on error
MOV   size_of_partial_save_array,AX     ;save size of partial save array
```

INT 67H

複数ハンドルページのマップ／アンマップ（論理ページ／物理ページ方式）

コール

AX　　＝5000H
DX　　＝EMMハンドル
CX　　＝配列内のエントリ数
DS：SI ＝配列のストラクチャへのポインタ

```
log_to_phys_map_struct          STRUC
  log_page_number               DW ?
  phys_page_number              DW ?
log_to_phys_map_struct          ENDS
```

リターン

AH ＝00H　**正常実行**（マップ完了）
　＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝83H　**回復不可**（指定のEMMハンドルがない）
　＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　＝8AH　**復旧可能**（マップされた論理ページがEMMハンドルにアロケートされている論理ページの範囲を超えている）
　＝8BH　**復旧可能**（物理ページがマップ可能な物理ページの範囲を超えている）
　＝8FH　**回復不可**（ファンクションパラメータが無効）

解　説

指定された複数の論理ページを、指定の物理ページにマッピングします。または指定の物理ページをアンマッピングします。

各ストラクチャのメンバは次のようになります。

.log_page_number

このメンバはワードで、マッピングされる論理ページ番号を示します。論理ページの番号は、0から〔EMMハンドルにアロケートされている論理ページの最大数–1〕までの範囲になります。もし論理ページがFFFFHにセットされていれば、指定の物理ページはアンマッピングされ、物理ページを介して論理ページへのアクセス（読み書き）はできなくなります。

.phys_page_number

2番目のメンバはワードで、論理ページがマッピングされる物理ページの番号を示します。物理ページの番号は0から〔システムがサポートしている物理ページの最大数–1〕までの範囲になります。

サンプル

```
log_to_phys_map            log_to_phys_map_struct? DUP(?)
emm_handle                 DW ?

MOV   AX,SEG log_to_phys_map        ;set log to phys map segment
MOV   DS,AX                         ;
LEA   SI,log_to_phys_map            ;DS:SI log to phys map point
MOV   CX,LENGTH log_to_phys_map     ;set arry entry
MOV   DX,emm_handle                 ;set handle
MOV   AX,5000H                      ;load function code
INT   67H                           ;call the memory manager
OR    AH,AH                         ;check EMM status
JNZ   EMM_ERR_HANDLER               ;jump to error handler on error
```

INT 67H

複数ハンドルページのマップ／アンマップ（論理ページ／セグメントアドレス方式）

コール

AX　＝5001H
DX　＝EMMハンドル
CX　＝配列内のエントリ数
DS：SI　＝配列のストラクチャへのポインタ

```
log_to_seg_map_struct            STRUC
  log_page_number                DW ?
  mappable_segment_address       DW ?
log_to_seg_map_struct            ENDS
```

リターン

AH ＝00H　**正常実行**（マップ完了）
　＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝83H　**回復不可**（指定のEMMハンドルがない）
　＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　＝8AH　**復旧可能**（マップされた論理ページがEMMハンドルにアロケートされている論理ページの範囲を超えている）
　＝8BH　**復旧可能**（指定されたマップ可能なセグメントアドレスがマップ不能）
　＝8FH　**回復不可**（ファンクションパラメータが無効）

解　　説

指定された複数の論理ページを指定のセグメントアドレスにマッピングします。または指定の物理ページをアンマッピングします。

各ストラクチャのメンバは次のようになります。

.log_page_number

このメンバはワードで、マッピングされる論理ページ番号を示します。論理ページの番号は、0から〔EMMハンドルにアロケートされている論理ページの最大数-1〕までの範囲になります。もし論理ページがFFFFHにセットされていれば指定の物理ページはアンマッピングされ、物理ページを介して論理ページへのアクセス（読み書き）はできなくなります。

.mappable_segment_addres
2番目のメンバはワードで、論理ページがマッピングされるセグメントアドレスを示します。このセグメントアドレスはマッピング可能なセグメントアドレスです。マッピング可能なセグメントアドレスは、ファンクション5800H（マップ可能な物理アドレス配列の取得）により知ることができます。

サンプル

```
log_to_phys_map                     log_to_phys_map_struct? DUP(?)
emm_handle                          DW ?

MOV   AX,SEG log_to_phys_map        ;set log to phys map segment
MOV   DS,AX                         ;
LEA   SI,log_to_phys_map            ;DS：SI log to phys map point
MOV   CX,LENGTH log_to_phys_map     ;set array entry
MOV   DX,emm_handle                 ;set handle
MOV   AX,5001H                      ;load function code
INT   67H                           ;call the memory manager
OR    AH,AH                         ;check EMM status
JNZ   EMM_ERR_HANDLER               ;jump to error handler on error
```

参　考

ファンクション5000H、5001H（論理ページ／物理ページ方式、論理ページ／セグメントアドレス方式）の1回のコールでシステムがサポートしている物理ページと、同数の論理ページをマッピング（アンマッピング）することができます。したがって、1度に1つのページをマッピングするより少ない処理時間で済みます。多くのページをマッピングするアプリケーションでは、このファンクションを使用してマッピングを行うと効率的です。

ここでは複数ページのマッピング、アンマッピングの方法について説明します。

・複数ページのマップ

このファンクションに渡されたEMMハンドルは、どのタイプの論理ページがマッピングされたかを決定します。ファンクション43H（ページのアロケート）やファンクション5A00H（標準サイズのページのアロケートと固有のEMMハンドルの割り当て）でアロケートされた論理ページはページとして参照され、そのサイズは16Kバイトです。ファンクション5A01H（ローページのアロケートと固有のEMMハンドルの割り当て）でアロケートされた論理ページはローページとして参照されますが、ファンクション43Hでアロケートされたページと同サイズとは限りません。

・複数ページのアンマップ

このファンクションは指定の物理ページを介した論理ページの読み書きを不可にすることができます。指定の物理ページからアンマッピングされた論理ページは、その物理ページから読み書きすることはできません。アンマッピングされた論理ページは再度同じ場所にマッピングするか、またはアンマッピング状態の物

理ページにマッピングすることで再び使用可能になります。物理ページのアンマッピングは論理ページFFFFHにセットすることにより完了します。

・複数ページのマップとアンマップの並行実行

ページのマッピングとアンマッピングは一度のコールで実行することができます。

マッピングまたはアンマッピング対象のページが存在しなくてもエラーにはなりません。たとえば、0ページのマッピングまたはアンマッピングの要求が実行されても何も実行されず、何のエラーも返しません。

・マップとアンマップ方法の変更

ページのマッピング、アンマッピングには次の2つの方法があります。どちらの方法でも結果は同じになります。

1. 論理ページとそれがマッピングされる物理ページを指定します。この方法はファンクション44H（ハンドルページのマップ／アンマップ）の拡張です。

2. 論理ページとそれがマッピングされるセグメントアドレスを指定します。これは基本的には1の方法と同じですが、ページの相対位置を表しているだけの番号を使用するよりも、物理ページの実際のアドレスを使用した方がより簡単です。メモリマネージャは、指定のセグメントアドレスがマッピング可能な物理ページの範囲内にあるかどうかをチェックします。そしてマネージャは渡されたセグメントアドレスを、ページにマッピングするために必要な内部表現に置き換えます。

INT 67H

ファンクション
51H ページの再アロケート

コール

AH = 51H
DX = EMM ハンドル
BX = reallocation_count
このファンクションがコールされた後で、このハンドルがアロケートするページの総数。

リターン

AH = 00H **正常実行**（再アロケート完了）
= 80H **回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H **回復不可**（拡張メモリハードウェアが動作不能）
= 83H **回復不可**（指定の EMM ハンドルがない）
= 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 87H **復旧可能**（システム内の使用可能なページ数は新しい再アロケーション要求にとって意味をもたない。プログラムは、より少ないページを EMM にアロケートするよう指定することによって復旧できる）
= 88H **復旧可能**（未アロケートページの数が新しいアロケーション要求にとって意味をもたない。プログラムは追加ページが使用可能なときに再度要求を出すか、より少ないページを指定することによって復旧できる）

BX =**再アロケーション後ハンドルにアロケートされたページ数**
ページが追加または削除された後、EMM ハンドルに現在アロケートされているページ数を示します。AH に 0 が返ってきた場合は、BX の値はこのファンクションの呼び出し前のハンドルにアロケートされたページ数と等しくなります。この情報は要求どおりの結果が得られたかどうかを調べる手掛かりとなります。

解　説

ページの再アロケートを行います。

このファンクションによって、EMM ハンドルにアロケートされている論理ページの数を増減することができます。BX に指定する再アロケーション数には次の 4 つの場合があります。

1. 再アロケーション数＝0
アプリケーションに割り当てられたハンドルはそのままで、アプリケーションがこれを使用することができます。メモリマネージャはハンドルを別のアプリケーションに再び割り当てることはしません。しかしハンドルはメモリマネージャに返し、アロケートされたすべてのページも保持しています。アプリケーションは、DOS に戻る前にファンクション 45H（ページのデアロケート（開放））を呼び出さなければなりません。すると、ハンドルは割り当てられたままで、別のアプリケーションを使用することはできません。

2. 再アロケーション数＝カレントのアロケーション数
これはエラーとしては扱いません。成功ステータス（AH = 0）を返します。

3. 再アロケーション数＞カレントのアロケーション数
メモリマネージャは、指定された EMM ハンドルのすでにアロケートされているページに、新たなページを増やそうとします。加えられた新たなページ数は、再アロケート数と現在のアロケート数の差です。EMM ハンドルにアロケートされていた論理ページの順番は、この操作後も変わりません。新たにアロケートされたページは、前にアロケートされたページが終わったところから、昇順に始まる論理ページ番号が付けられます。

4. 再アロケーション数＜カレントのアロケーション数
メモリマネージャは、現在アロケートされたページのいくつかを取り除き、それらをメモリマネージャに返そうとします。取り除かれるページの数は、現在のアロケート数と再アロケート数の差です。ページは指定された EMM ハンドルの現在アロケートされているページ列の最後から取り除かれます。EMM ハンドルにアロケートされた論理ページの順番は、この操作後も変わりません。
どのような型の論理ページが再アロケートされるかは EMM ハンドルで決まります。ファンクション 43H でアロケートされた論理ページはページと呼ばれ、16K バイトの大きさを持ちます。ファンクション 5A00H でアロケートされた論理ページはロー（raw）ページと呼ばれ、ファンクション 43H でアロケートしたページの大きさと同じとは限りません。

サンプル

```
emm_handle                          DW ?
realloc_count                       DW ?
current_alloc_page_count            DW ?

MOV   DX,emm_handle                 ;set emm handle
MOV   BX,realloc_count              ;set relloc count
MOV   AH,51H                        ;load function code
INT   67H                           ;call the memory manager
OR    AH,AH                         ;check EMM status
JNZ   EMM_ERR_HANDLER               ;jump to error handler on error
MOV   current_alloc_page_count,BX   ;save current alloc page count
```

INT 67H

ハンドルアトリビュートの取得

コール

AX ＝5200H
DX ＝EMMハンドル

リターン

AH ＝00H　**正常実行**（EMMハンドルのアトリビュートを取得した）
＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
＝81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝83H　**回復不可**（指定のEMMハンドルがない）
＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝8FH　**回復不可**（ファンクションパラメータが無効）
＝91H　**回復不可**（サポートされていない）
AL ＝handle_attribute
EMMハンドルのアトリビュートを示します。
0　ハンドルは揮発性
1　ハンドルは不揮発性

解　説

ハンドルに関するアトリビュートを返します。

アトリビュートは揮発性か不揮発性のどちらかです。不揮発性のアトリビュートのハンドルはメモリマネージャがウォームブートの間、ハンドルページの内容をセーブできるようにします。しかし、このファンクションはユーザーオプションで使えない場合や、メモリボードやシステムハードウェアでサポートされていない場合があります。

ハンドルのアトリビュートが不揮発性にセットされていると、ハンドルとその名前（割り当てられている場合)、ハンドルにアロケートされたページの内容は、ウォームブート後もすべて保持されています。

注意　PC-9800シリーズでは、揮発性のアトリビュートのみ通知されます。

サンプル

```
emm_handle                              DW ?
handle_attribute                        DB ?

MOV    DX,emm_handle                    ;set emm handle
MOV    AX,5200H                         ;load function code
```

```
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
MOV   handle_attribute,AL       ;save handle attribute
```

INT 67H

ファンクション
5201H　ハンドルアトリビュートの設定

コール

AX ＝ 5201H
DX ＝ EMMハンドル
BL ＝ new_handle_attribute
EMMハンドルの新しいアトリビュートを示します。
0　EMMハンドルは揮発性
1　EMMハンドルは不揮発性
揮発性のEMMハンドルアトリビュートは、メモリマネージャにウォームブートした後、EMMハンドルにアロケートされているページとEMMハンドルの両方をデアロケートするように知らせます。もし、全EMMハンドルが揮発性のアトリビュート（既定のアトリビュート）ならば、EMMハンドルのディレクトリは空になり拡張メモリすべてがウォームブート後すぐに0に初期化されます。

リターン

AH ＝ 00H　**正常実行**（EMMハンドルのアトリビュートを設定した）
＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H　**回復不可**（指定のEMMハンドルがない）
＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8FH　**回復不可**（ファンクションパラメータが無効）
＝ 90H　**回復不可**（指定のアトリビュートが未定義）
＝ 91H　**回復不可**（サポートされていない）

解　説

EMMハンドルに関するアトリビュートを修正します。

EMMハンドルが持つアトリビュートは揮発性または不揮発性です。不揮発性アトリビュートは、EMMがウォームブートの間のハンドルのページ内容を保持できるようにします。しかし、このファンクションはユーザーオプションで使用できない場合や、メモリボードやシステムハードウェアでサポートされていない場合があります。もしEMMハンドルのアトリビュートが不揮発性にセットされていれば、EMMハンドルまたはEMMハンドルの名前（割り当てられている場合）、EMMハンドルにアロケートされているページの内容はウォームブート後もすべて保持されます。

注意　PC-9800シリーズでは、不揮発性のアトリビュートをサポートしていません。不揮発性を指定した場合には、AH = 91Hが返されます。

サンプル

```
emm_handle                          DW ?
new_handle_attrib                   DB ?

MOV    DX,emm_handle                ;set emm handle
MOV    BX,new_handle_attrib         ;set new handle attribute
MOV    AX,5201H                     ;load function code
INT    67H                          ;call the memory manager
OR     AH,AH                        ;check EMM status
JNZ    EMM_ERR_HANDLER              ;jump to error handler on error
```

INT 67H

ハンドルアトリビュートのケイパビリティの取得

コール AX = 5202H

リターン AH = 00H 正常実行（EMMハンドルのアトリビュートを取得した）
= 80H 回復不可（メモリマネージャソフトウェアが動作不能）
= 81H 回復不可（拡張メモリハードウェアが動作不能）
= 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）
= 8FH 回復不可（ファンクションパラメータが無効）
AL = attribute_capability
アトリビュートの性能を示します。
0 メモリマネージャとハードウェアは揮発性のEMMハンドルのみをサポート
1 メモリマネージャとハードウェアは不揮発性のEMMハンドルと揮発性のEMMハンドルの両方をサポート

解　　説

メモリマネージャが不揮発性のアトリビュートをサポートできるかを調べます。

注意　PC-9800シリーズでは、常に揮発性のアトリビュート（AL = 0）を返します。

サンプル

```
attrib_capability              DB ?

MOV   AX,5202H                 ;load function code
INT   67H                      ;call the memory manager
OR    AH,AH                    ;check EMM status
JNZ   EMM_ERR_HANDLER          ;jump to error handler on error
MOV   attrib_capability,AL     ;save attribute capability
```

INT 67H

ファンクション 5300H　ハンドル名の取得

コール

AX　　＝5300H
DX　　＝EMMのハンドル番号
ES：DI ＝handle_name 配列
　8バイトの配列で、現在のEMMハンドルに割り当てられている名前がコピーされます。

リターン

AH ＝00H　正常実行（ハンドル名を取得した）
　＝80H　回復不可（メモリマネージャソフトウェアが動作不能）
　＝81H　回復不可（拡張メモリハードウェアが動作不能）
　＝83H　回復不可（指定のEMMハンドルがない）
　＝84H　回復不可（メモリマネージャに渡されたファンクションが未定義）
　＝8FH　回復不可（ファンクションパラメータが無効）
handle_name 配列
　指定のEMMハンドルの名前も含みます。

解　説

現在EMMハンドルに割り当てられている8文字の名前を取得します。

ハンドル名に使用されるキャラクタについては制限はありません（ASCIIコード0〜FFH）。ハンドル名はNULL（00H）に3回初期化されます。これはメモリマネージャがインストールされたとき、ハンドルがアロケートされるとき、EMMハンドルがデアロケートされるときの3回です。ハンドル名がすべてNULLならば、名前はついていないと見なされます。EMMハンドルに名前を割り当てるとき、名前のないEMMハンドルと区別するために少なくとも名前の1文字はNULLでないキャラクタでなければなりません。

サンプル

```
handle_name                     DB 8  DUP(?)
emm_handle                      DW ?

MOV    AX,SEG handle_name       ;set handle name segment
MOV    ES,AX                    ;
LEA    DI,handle_name           ;ES：DI handle name pointer
MOV    DX,emm_handle            ;set emm handle
```

```
MOV   AX,5300H              ;load function code
INT   67H                   ;call the memory manager
OR    AH,AH                 ;check EMM status
JNZ   EMM_ERR_HANDLER       ;jump to error handler on error
```

INT 67H

ファンクション 5301H ハンドル名の設定

コール

AX　　= 5301H
DX　　= EMMハンドル番号
DS：SI = handle_nameへのポインタ
　EMMハンドルに割り当てられる名前を含むバイトの配列。ハンドル名が8バイトに満たない場合は残りをNULLで埋めます。

リターン

AH = 00H　**正常実行**（ハンドル名を設定した）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 83H　**回復不可**（指定のEMMハンドルがない）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（ファンクションパラメータが無効）
= A1H　**復旧可能**（この名前を持つハンドルはすでに存在する。指定のハンドルに名前の割り当てができない）

解説

EMMハンドルに対し8文字の名前を割り当てます。ハンドル名に使用されるキャラクタについては制限はありません。全文字（ASCIIコード0～FFH）を名前の各キャラクタとして割り当てることができます。

インストール時、すべてのEMMハンドルにはNULL（00H）で初期化された名前がつけられています。名前の中が全部NULLならば名前はついていません。EMMハンドルに名前を割り当てるとき、少なくとも1文字は名前のないEMMハンドルと区別するために、NULLではないASCIIキャラクタでなければなりません。また同じ名前を他のEMMハンドルが持つことはできません。

EMMハンドルは、EMMハンドルに新しい値をセットすることで名前を変更することができます。またハンドル名を全部NULLにセットすると、名前を削除することもできます。EMMハンドルがデアロケートされると、名前はなくなります。

サンプル

```
handle_name                          DB "AARDVARK"
emm_handle                           DW ?

MOV    AX,SEG handle_name            ;set handle name segment
```

```
MOV   DS,AX               ;
LEA   SI,handle_name      ;DS:SI handle name pointer
MOV   DX,emm_handle       ;set emm handle
MOV   AX,5301H            ;load function code
INT   67H                 ;call the memory manager
OR    AH,AH               ;check EMM status
JNZ   EMM_ERR_HANDLER     ;jump to error handler on error
```

INT 67H

ファンクション 5400H　ハンドルのディレクトリ取得

コール

AX　= 5400H
ES：DI = handle_dir へのポインタ
　メモリマネージャがハンドルのディレクトリをコピーするメモリエリアへのポインタ。

```
Handle_dir_struct        STRUC
  handle_value           DW ?
  handle_name            DB 8 DUP (?)
Handle_dir_struct        ENDS
```

リターン

AH = 00H　**正常実行**（ハンドルのディレクトリを取得した）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 83H　**回復不可**（指定の EMM ハンドルがない）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8FH　**回復不可**（ファンクションパラメータが無効）

handle_dir
　EMM ハンドルの値と各 EMM ハンドルに関連するハンドル名を含みます。

AL = handle_dir **配列のエントリ数**
　ハンドルのディレクトリ配列内のエントリ数を示します。これはオープンな EMM ハンドルの数と同じです。たとえば、オープンな EMM ハンドルが 1 つであれば AL には 1 が返されます。

解　説

すべてのアクティブな EMM ハンドルと、各ハンドルに割り当てられている名前を返します。名前を割り当てられていない EMM ハンドルは、ASCII の NULL（バイナリで 0）で埋まったデフォルト名を持ちます。EMM ハンドルが最初にアロケートされたとき、または EMM ハンドルに属するすべてのページがデアロケートされたとき（つまりオープン EMM ハンドルがクローズされたとき）は、EMM ハンドルのデフォルト名は NULL にセットされます。これは後で EMM ハンドルがオープンされたとき、名前を持てるようにするためです。名前に割り当てられる値は ASCII コードの 0～FFH です。

配列に要求されるバイト数は 10 バイト× EMM ハンドルの合計数（1 エントリ当り 10 バイト）でこの配列の大きさの上限は 10 バイト× 255 = 2550 バイトです。

また、各ストラクチャのメンバは次のようになります。

.handle_value
最初のメンバはワードで、オープンなEMMハンドルを示します。

.handle_name
2番目のメンバはEMMハンドルのASCII名を含む8バイトの配列です。現在ハンドルに名前がなければ、すべてNULLがセットされています。

サンプル

```
handle_dir                        handle_dir_struct 255 DUP(?)
num_entries_in_handle_dir         DB ?

MOV    AX,SEG handle_dir          ;set handle dir segment
MOV    ES,AX                      ;
LEA    DI,handle_dir              ;ES:DI handle dir pointer
MOV    AX,5400H                   ;load function code
INT    67H                        ;call the memory manager
OR     AH,AH                      ;check EMM status
JNZ    EMM_ERR_HANDLER            ;jump to error handler on error
MOV    num_entries_in_handle_dir,AL
                                  ;save number of entries in handle dir
```

INT 67H

ファンクション 5401H 指定ハンドルのサーチ

コール

AX　　= 5401H
DS：SI = handle_name
　サーチする名前を含む8バイトの文字列へのポインタ。

リターン

AH = 00H　正常実行（指定の名前のついたEMMハンドルが見つかった）
　= 80H　回復不可（メモリマネージャソフトウェアが動作不能）
　= 81H　回復不可（拡張メモリハードウェアが動作不能）
　= 84H　回復不可（メモリマネージャに渡されたファンクションが未定義）
　= 8FH　回復不可（ファンクションパラメータが無効）
　= A0H　復旧可能（EMMハンドルが見つからない）
　= A1H　回復不可（ハンドル名がない。全部NULLコード）
DX =指定した名前と一致したハンドルの値

解説

固有の名前をもつハンドルを、ハンドル名のディレクトリからサーチします。

もしその名前のついたハンドルが見つかれば、その名前のハンドル番号を返します。すべてのハンドルはインストール時にその名前をすべてNULLにセットします。ハンドル名がすべてNULLならば、名前はついていないと見なされます。ハンドルに名前を割り当てるとき、名前のないハンドルと区別するために少なくとも名前の1文字はNULLでないキャラクタでなければなりません。

サンプル

```
named_handle                    DB 'AARDVARK'
named_handle_value              DW ?

MOV   AX,SEG named_handle       ;set named handle segment
MOV   DS,AX                     ;
LEA   SI,named_handle           ;DS：SI  named handle pointer
MOV   AX,5401H                  ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
MOV   named_handle_value,DX     ;save named handle value
```

INT 67H

ハンドルの総数の取得

コール　AX ＝ 5402H

リターン　AH ＝ 00H　**正常実行**（サポートされているハンドルの総数を返した）
＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8FH　**回復不可**（ファンクションパラメータが無効）
BX ＝ total_handles
この値はプログラムがメモリマネージャにアロケート要求の出せるハンドルの最大数を示します。また、この値はOS用のハンドル（ハンドル0）も含みます。

解　説

OSのEMMハンドル（ハンドル0）を含め、メモリマネージャがサポートしているEMMハンドルの総数を返します。

サンプル

```
total_handles                       DW ?

MOV   AX,5402H                      ;load function code
INT   67H                           ;call the memory manager
OR    AH,AH                         ;check EMM status
JNZ   EMM_ERR_HANDLER               ;jump to error handler on error
MOV   total_handles,BX              ;save total handles
```

INT 67H

ファンクション 55H ページマップの変更とジャンプ

コール

AH　　= 55H

AL　　= physical page number/segment selector

log_phy_map ストラクチャ内の phys_page_number_seg メンバの値が物理ページ番号を表しているセグメントなのか、物理ページ番号なのかを示すコード。

AL = 0　物理ページ番号

AL = 1　物理ページ番号のセグメントアドレス

DX　　= EMM ハンドル番号

DS：SI = map_and_jump ストラクチャへのポインタ

要求された物理ページをマッピングし、ターゲットアドレスにジャンプするために必要な情報を含むストラクチャへのポインタ。

```
log_phys_map_struct             STRUC
  log_page_number               DW ?
  phys_page_number_seg          DW ?
log_phys_map_struct             ENDS

map_and_jumo_struct             STRUC
  target_address                DD ?
  log_phys_map_len              DB ?
  log_phys_mappty               DD ?
map_and_jump_struct             ENDS
```

リターン

AH = 00H　**正常実行**（指定のエリアに制御が渡った）

= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）

= 81H　**回復不可**（拡張メモリハードウェアが動作不能）

= 83H　**回復不可**（指定の EMM ハンドルがない）

= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）

= 8AH　**復旧可能**（マップされた論理ページが EMM ハンドルにアロケートされている論理ページの範囲を超えている）

= 8BH　**復旧可能**（指定されたマップ可能なセグメントアドレスがマップ不能）

= 8FH　**回復不可**（ファンクションパラメータが無効）

解　　説

メモリへのマッピングを変更し、指定のアドレスに制御を渡します。これは8086のFAR JUMP機能と同じようなものです。このファンクションをコールすると、コールされる前のメモリマッピングコンテキストは消えます。

ページをマッピングしないでジャンプしてもエラーにはなりません。もし、ページをマッピングしないでジャンプする要求があった場合は、目的のアドレスに制御が渡り、このファンクションはFAR JUMPを実行します。

また、各ストラクチャのメンバは次のようになります。

.target_address

最初のメンバは制御が移されるターゲットアドレスを含むFARポインタです。アドレスはセグメント：オフセット形式で表現されます。アドレスのオフセット値はダブルワードの下位ワードに格納されます。

.log_phys_map_len

2番目のメンバはバイトで、すぐ後に続くストラクチャの配列中のエントリ数を示します。配列の大きさは、要求された論理ページを物理ページにマッピングするために必要な長さです。エントリの数は、システム中でマッピングできる物理ページの数を超えることはできません。

.log_phys_map_ptr

3番目のメンバは、論理ページ番号とマッピングされる物理ページまたはセグメントアドレスを含んでいるストラクチャの配列へのポインタです。ストラクチャの配列の各エントリは次の2つです。

.log_page_number

このストラクチャの最初のメンバは、マッピングされる論理ページの数を含むワードです。

.phys_page_number_seg

このストラクチャの2番目のメンバはワードで、最初のメンバの論理ページがマッピングされる物理ページの番号か、またはセグメントアドレス形式のどちらかを含みます。ALに渡される値で、どちらの表現かが決定されます。

サンプル

```
log_phys_map                    log_phys_map_struct(?)DUP(?)
map_and_jump                    map_and_jump_struct(?)
emm_handle                      DW ?
phys_page_or_seg_mode           DB ?

MOV    AX,SEG map_and_jump      ;set map and jump segment
MOV    DS,AX                    ;
LEA    SI,map_and_jump          ;DS：SI map and jump pointer
```

55H

```
MOV   DX,emm_handle                ;set EMM handle
MOV   AH,55H                       ;load function code
MOV   AL,phys_page_or_seg_mode     ;set phys page or seg mode
INT   67H                          ;call the memory manager
OR    AH,AH                        ;check EMM status
JNZ   EMM_ERR_HANDLER              ;jump to error handler on error
```

INT 67H

ファンクション
56H　ページマップの変更とコール

コール

AH　　= 56H
AL　　= physical page number/segment selector
　　log_phy_map ストラクチャ内の phys_page_number_seg メンバの値が物理ページ番号を表しているセグメントなのか、物理ページ番号なのかを示すコード。
　　AL = 0　物理ページ番号
　　AL = 1　物理ページ番号のセグメントアドレス
DX　　= EMM ハンドル番号
DS：SI = map_and_call ストラクチャへのポインタ
　　要求された論理ページを指定の物理ページにマッピングし、ターゲットアドレスをコールするために必要な情報を含むストラクチャへのポインタ。

```
log_phys_map_struct             STRUC
  log_page_number               DW ?
  phys_page_number_seg          DW ?
log_phys_map_struct             ENDS

map_and_call_struct             STRUC
  target_address                DD ?
  new_page_map_len              DB ?
  new_page_map_ptr              DD ?
  old_page_map_len              DB ?
  old_page_map_ptr              DD ?
  reserved                      DW 4DUP (?)
map_and_call_struct             ENDS
```

リターン

AH = 00H　**正常実行**（制御がターゲットアドレスに渡った）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 83H　**回復不可**（指定の EMM ハンドルがない）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8AH　**復旧可能**（対応する物理ページにマップされる1つ以上の論理ページが、EMM ハンドルにアロケートされている論理ページの範囲を超えた。プログラムは、EMM ハンドルの範囲内で論理ページをマッピングすることにより復旧できる）

55H/56H

＝8BH	**復旧可能**（1つ以上の物理ページが、指定できる物理ページの範囲を超えた。またはシステム内に存在する物理ページ数以上のページを指定した。このとき物理ページ番号は0から番号づけられる。プログラムは0から（システムがサポートしている物理ページ数–1）までの範囲内の物理ページをマッピングすることにより復旧できる）
＝8FH	**回復不可**（ファンクションパラメータが無効）

解　　説

カレントのマッピングコンテキストをセーブして、指定したマッピングコンテキストの変更を行って、指定のアドレスに制御を渡します。これは、8086のFAR CALLの機能と同じようなものです。FAR CALLから戻ったときにコードセグメント内の値を元に戻すのと同様に、このファンクションもリターン、指定のマッピングコンテキストを元の状態に戻します。

FAR CALLからのリターンを、そのままエミュレートするような拡張メモリファンクションはありませんが、FAR CALLの標準的なリターンを実行することできます。次にこの機能について説明します。

このファンクションの起動後にエラーが検出されなければ、メモリマネージャが指定されたアドレスに制御を移します。エラーが生じた場合は、メモリマネージャはただちにAHレジスタにエラーコードを返しますが、発生しなかった場合、メモリマネージャはリターン後にマッピングコンテキストの状態を復元するために、スタックに情報をプッシュしておきます。

呼び出されたプロシージャが、呼び出したプロシージャに値を返す必要がある場合は、標準的なFAR RETURNを行います。メモリマネージャはこのリターンとトラップし、退避したマッピングコンテキストを復元して、呼び出したプロシージャにリターンします。メモリマネージャは他のファンクションと同様に、成功した場合にもステータスを返します。このファンクションを用いる場合は、使用する分のスタックスペースを考慮しなければなりません。

また、各ストラクチャのメンバは次のようになります。

.target_address

最初のメンバは、制御が移されるターゲットアドレスを含むFARポインタです。アドレスはセグメント：オフセット形式で表現されます。アドレスのオフセット部分はポインタの下位ワードに格納されます。アプリケーションはこの値を供給しなければなりません。

.new_page_map_len

2番目のメンバはバイトで、new_page_map_ptrが指す新しいマッピングコンテキストのエントリ数を示します。この値はシステム内でマッピング可能なページ数を超えることはできません。

.new_page_map_ptr

3番目のメンバは、論理ページ番号とコール後にマッピングされる物理ページ番号、またはセグメントを含むストラクチャの配列へのFARポインタです。新しいストラクチャの配列内容はmap_and_call

ストラクチャの後に置かれます。

.old_page_map_len

4番目のメンバはバイトで、old_page_map_ptrが指す古いマッピングコンテキスト内のエントリ数を示します。この値はシステム内でマッピング可能なページ数を超えることはできません。

.old_page_map_ptr

5番目のメンバは、論理ページ番号とリターン後にマッピングされる物理ページ番号、またはセグメントを含むストラクチャの配列へのFARポインタです。元のストラクチャの配列内容はmap_and_callストラクチャの後に置かれます。

.reserved

6番目のメンバはメモリマネージャ用に予約されています。

ストラクチャの配列内の各エントリは次の2つのメンバです。

.log_page_number

このストラクチャの最初のメンバは、コールやリターン直後に2番目のメンバで指定される物理ページ番号、またはセグメントアドレス表現にマッピングする論理ページ番号を表すワードです。

.phys_page_number_seg

このストラクチャの第2のメンバは、コールやリターン直後に、最初のメンバで指定される論理ページ番号に、マッピングされる物理ページの番号かセグメントアドレス表現のどちらかを表すワードです。

サンプル

```
new_page_map            log_phys_map_struct(?)DUP(?)
old_page_map            log_phys_map_struct(?)DUP(?)
map_and_call            map_and_call_struct(?)DUP(?)
emm_handle              DW ?
phys_page_or_seg_mode   DB ?

MOV   AX,SEG map_and_jump         ;set map and jump segment
MOV   DS,AX                       ;
LEA   SI,map_and_call             ;DS：SI map and call pointer
MOV   DX,emm_handle               ;set EMM handle
MOV   AH,56H                      ;load function code
MOV   AL,phys_page_or_seg_mode    ;set phys page or seg mode
INT   67H                         ;call the memory manager
OR    AH,AH                       ;check EMM status
JNZ   EMM_ERR_HANDLER             ;jump to error handler on error
```

INT 67H

ファンクション
5602H　ページマップスタックサイズの取得

コール　AX ＝ 5602H

リターン　AH ＝ 00H　正常実行（スタックサイズが返された）
＝ 80H　回復不可（メモリマネージャやソフトウェアが動作不能）
＝ 81H　回復不可（拡張メモリハードウェアが動作不能）
＝ 84H　回復不可（メモリマネージャに渡されたファンクションが未定義）
＝ 8FH　回復不可（ファンクションパラメータが無効）
BX ＝要求されたスタックスペース

解　説

ファンクション56H（ページマップの変更とコール）で要求されるページマップスタックサイズを取得します。

ファンクション56H（ページマップの変更とコール）はスタック上に追加情報（リターンアドレスを含む）をプッシュします。このファンクションはそのファンクションが要求するスタックスペースのバイト数を返します。

INT 67H

メモリ領域の移動

コール

AX　　= 5700H

DS：SI = move_source_dest ストラクチャへのポインタ

移動するためのソースとデスティネーションの情報を含むストラクチャへのポインタ。

```
move_source_dest_struct          STRUC
  region_length                  DD ?
  source_memory_type             DB ?
  source_handle                  DW ?
  source_initial_offset          DW ?
  source_initial_seg_page        DW ?
  dest_memory_type               DB ?
  dest_handle                    DW ?
  dest_initial_offset            DW ?
  dest_initial_seg_page          DW ?
move_source_dest_struct          ENDS
```

リターン

AH = 00H　**正常実行**（メモリ領域の移動を行った）

= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）

= 81H　**回復不可**（拡張メモリハードウェアが動作不能）

= 83H　**復旧可能**（ソースまたはデスティネーションの EMM ハンドルが見つからなかった。メモリマネージャは指定の情報を得られなかった。EMM ハンドルは無効）

= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）

= 8AH　**回復不可**（指定の論理ページが EMM ハンドルに割り当てられているページを超えた）

= 8FH　**回復不可**（ファンクションパラメータが無効）

= 92H　**成功**（ソースまたはデスティネーションの拡張メモリ領域は、同じ EMM ハンドルを持ち重複している。この場合、移動は行える。移動は完了し、ソース領域は全部デスティネーションにコピーされた。しかし、ソース領域は移動によって上書きされている。異なるハンドルを持つソース領域とデスティネーション領域は、それぞれ異なるメモリ領域を指定するので、物理的には重複されることはない）

= 93H　**状況により復旧可能**（指定されたソースまたはデスティネーショ

5602H/5700H

ンの拡張メモリの大きさが、ソースまたはデスティネーションハンドルにアロケートされている拡張メモリページの大きさを超えた。このハンドルにアロケートされているページサイズでは、領域のコピーができない。プログラムは、ソースまたはデスティネーションハンドルに追加のページをアロケートし、再度このファンクションを実行することにより、上記の状況から復旧できる。ただし、アプリケーションが必要としているにもかかわらず、すでに拡張メモリをアロケートしていた場合、エラーとなり復旧できない）

＝94H　**回復不可**（内部メモリと拡張メモリの領域が重複している。これは無効で内部メモリは拡張メモリと重複することはできない）

＝95H　**回復不可**（論理ページ内のオフセットが論理ページの大きさを超えた。拡張メモリ領域内の最初のソースまたはデスティネーションのオフセットは、0000Hから3FFFH（16383）、または（論理ページの大きさ–1）でなくてはならない）

＝96H　**回復不可**（領域サイズが1Mバイトを超えた）

＝98H　**回復不可**（メモリのソースとデスティネーションのタイプが定義されていない。またはサポートされていない）

＝A2H　**回復不可**（移動中に内部メモリの1Mバイトアドレススペースを超えようとした。ソース／デスティネーションの開始アドレスの組み合わせと、移動する領域の大きさが1Mバイトを超えた。データは移動されなかった）

解　説

メモリ領域の移動を行います。

メモリ領域の移動は、メモリのソース／デスティネーションの組み合わせで次のようになります。

- 内部メモリ→内部メモリ
- 内部メモリ→拡張メモリ
- 拡張メモリ→内部メモリ
- 拡張メモリ→拡張メモリ

メモリを移動するために、拡張メモリのマッピングコンテキストを保存、復元する必要はありません。カレントのマッピングコンテキストは、このオペレーションを行っても保持されます。

領域の大きさは、指定のEMMハンドルにアロケートされた拡張メモリページのサイズによって制限されます。しかし、ほとんどのアプリケーションでは領域の大きさは限界サイズよりも小さくなっています。領域サイズが0でもエラーではなく、メモリの移動は実行されません。

領域サイズが16Kバイトを超えてもエラーになりません。この場合、論理ページの1グループが移動

の対象になります。指定の論理ページは、移動が行われる最初の論理ページを表します。もし領域のサイズが16Kバイトを超えた場合、または16Kバイト未満であっても複数の論理ページにまたがっていれば、全体の領域に合わせるために最初の論理ページの後に、十分な大きさの論理ページが残っていなくてはなりません。

アプリケーションが拡張メモリに内部メモリ領域をセーブする必要があれば、マッピングコンテキストの保存、または復元を実行しなくても領域を移動することができます。メモリマネージャはこのコンテキストを維持し、最大1Mバイトまでの移動ができます。しかし、実際の移動サイズはこれよりも小さくなります。

もし、ソースEMMハンドルとデスティネーションEMMハンドルが同じ場合、ソース領域とデスティネーション領域は、移動の前に重複しているかどうかチェックされます。そして領域が重複していても移動方向が正しく選択され、デスティネーション領域にはソース領域が完全にコピーされ、領域の重複が発生したことを示すステータスが返されます。

各ストラクチャのメンバは次のようになります。

.region_length

最初のメンバはダブルワードで、移動されるメモリ領域の大きさ（バイト）を指定します。

.source_memory_type

2番目のメンバはバイトで、ソース領域のメモリのタイプを指定します。これが0ならば、ソース領域は内部メモリ内（ページフレームセグメントを除く）に存在し、1ならば拡張メモリ内に存在することを示します。

.source_handle

3番目のメンバはワードで、ソース領域が拡張メモリ内にある場合、ソースメモリ領域と関連するEMMハンドル番号を指定します。また、ソース領域が内部メモリ内にある場合は、この変数は意味がなく将来の互換性のために0にセットしなければなりません。

.source_initial_offset

4番目のメンバはワードで、移動を開始するソース領域内のオフセットを指定します。ソース領域が拡張メモリ内にある場合、source_initial_offsetは16Kバイトの論理ページの最初のアドレスを基準にするので、オフセットは0000Hから3FFFHまでの値をとります。また、ソース領域が内部メモリ内にある場合、source_initial_offsetは移動を開始するソースセグメントの最初のオフセットを指定します。このオフセットは64Kバイトの内部メモリの最初のアドレスを基準にしているため、この値は0000HからFFFFHまでの値になります。

.source_initial_seg_page

5番目のメンバはバイトで、移動を開始するソース領域のセグメント、または論理ページ番号を指定します。ソース領域が拡張メモリ内にある場合、移動を開始するソース領域の論理ページを指定します。また、ソース領域が内部メモリ内にある場合、source_initial_seg_pageは、移動を開始する内部メモリの最初のセグメントアドレスを指定します。

.dest_memory_type

6番目のメンバはバイトで、デスティネーション領域が存在するメモリのタイプを指定します。0ならば内部メモリで、1ならば拡張メモリに存在することを示します。

.dest_handle

7番目のメンバはワードで、デスティネーション領域が拡張メモリ内にある場合、デスティネーションメモリ領域に関連するEMMハンドル番号を指定します。デスティネーション領域が内部メモリ内にある場合、この変数は意味がなく将来の互換性のために0にセットしなければなりません。

.dest_initial_offset

8番目のメンバはワードで、移動を開始するデスティネーション領域内のオフセットを指定します。デスティネーション領域が拡張メモリ内にある場合、dest_initial offsetは16Kバイトの論理ページの最初のアドレスを基準にして、オフセットは0000Hから3FFFHまでの値をとります。また、デスティネーション領域が内部メモリ内にある場合、dest_initial_offsetは移動を開始するデスティネーションセグメントの、最初のオフセットを指定します。このオフセットは64Kバイトの内部メモリの最初のアドレスを基準にしているため、この値は0000HからFFFFHまでの値をとります。

.dest_initial_seg_page

9番目のメンバーはワードで、移動を開始するデスティネーション領域のセグメントまたは論理ページ番号を指定します。デスティネーション領域が拡張メモリ内にある場合、移動を開始するデスティネーション領域の論理ページを指定します。また、デスティネーション領域が内部メモリ内にある場合、dest_initial_seg_pageは移動を開始する内部メモリの最初のセグメントアドレスを指定します。

サンプル

```
move_source_dest                move_source_dest_struct(?)

MOV   AX,SEG move_source_dest   ;set move source dest segment
MOV   DS,AX                     ;
LEA   SI,move_source_dest       ;DS:SI move source dest pointer
MOV   AX,5700H                  ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
```

INT 67H

メモリ領域の交換

コール

AX = 5701H
DS：SI = exchange_source_dest ストラクチャへのポインタ
交換するためのソースとデスティネーションの情報を含む、ストラクチャへのポインタ。

```
xchg_source_dest_struct          STRUC
  region_length                  DD ?
  source_memory_type             DB ?
  source_handle                  DW ?
  source_initial_offset          DW ?
  source_initial_seg_page        DW ?
  dest_memory_type               DB ?
  dest_handle                    DW ?
  dest_initial_offset            DW ?
  dest_initial_seg_page          DW ?
xchg_source_dest_struct          ENDS
```

リターン

AH = 00H **正常実行**（メモリ領域の交換が行われた）
= 80H **回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H **回復不可**（拡張メモリハードウェアが動作不能）
= 83H **復旧可能**（ソースまたはデスティネーションの EMM ハンドルが見つからなかった。メモリマネージャは指定のハンドルの情報を得られなかった。EMM ハンドルは無効）
= 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8AH **回復不可**（EMM ハンドルに割り当てられているページ数より大きいページが指定された）
= 8FH **回復不可**（ファンクションパラメータが無効）
= 93H **状況により復旧可能**（指定されたソースまたはデスティネーションの拡張メモリの大きさが、ソースまたはデスティネーションハンドルにアロケートされている拡張メモリページの大きさを超えた。このハンドルにアロケートされているページサイズでは、領域のコピーができない。プログラムはソースまたはデスティネーションハンドルに追加のページをアロケートし、再度このファンクションを実行することにより、この状況から復旧できる。しかし、アプリケーションが必要としているにもかかわらず、すでに拡

張メモリがアロケートされていれば、エラーになり復旧できない）

＝94H	**回復不可**（内部メモリと拡張メモリの領域が重複している。これは無効で、内部メモリは拡張メモリと重複することはできない）
＝95H	**回復不可**（論理ページ内のオフセットが、論理ページの大きさを超えた。拡張メモリ領域内の最初のソースまたはデスティネーションのオフセットは、0000Hから3FFFH（16383）、または〔論理ページの大きさ–1〕でなくてはならない）
＝96H	**回復不可**（領域サイズが1Mバイトを超えた）
＝97H	**回復不可**（ソースとデスティネーションの拡張メモリ領域が同じハンドルを持ち、領域が重複している。これは無効で、ソースとデスティネーションの拡張メモリ領域が交換される場合は、同じハンドルを持つため重複することはできない。異なるハンドルを持つソースとデスティネーションの拡張メモリ領域は、普通異なる拡張メモリ領域を指定するので物理的には決して重複することはない）
＝98H	**回復不可**（メモリのソースとデスティネーションのタイプが定義されていない。またはサポートされていない）
＝A2H	**回復不可**（移動中に内部メモリの1Mバイトアドレススペースを超えようとした。ソース／デスティネーション開始アドレスの組み合わせと、交換されるべき領域の大きさが1Mバイトを超えた。データは交換されなかった）

解　説

メモリ領域の交換を行います。

メモリ領域の交換（ストリング転送）は、メモリのソース／デスティネーションの組み合わせで次のようになります。

- 内部メモリ→内部メモリ
- 内部メモリ→拡張メモリ
- 拡張メモリ→内部メモリ
- 拡張メモリ→拡張メモリ

拡張メモリ領域は、640Kバイト（9FFFFH）以上のメモリエリアのみを指します（PC–9800シリーズでは1Mバイト（100000H）以上を指します）。もしシステムがマッピング可能な内部メモリを備えていれば、このファンクションはマッピング可能な内部メモリを、普通の内部メモリとして扱います。ソース領域の内容とデスティネーション領域の内容は交換されます。

交換のオペレーションを行うために、拡張メモリのマッピングコンテキストを保存、復元する必要はありません。カレントのマッピングコンテキストは、このオペレーションの間保持されます。領域の大き

さは、指定の EMM ハンドルにアロケートされている拡張メモリページのサイズにより制限されます。また、領域の大きさが 0 でもエラーにならず交換も実行されません。16K バイトを超える領域サイズもエラーにはなりません。この場合、このファンクションは論理ページの 1 グループが交換の対象になります。

指定された論理ページは、交換が実行される最初の論理ページを表しています。

もし、領域の大きさが 16K バイトを超えるとき、または領域のサイズが 16K バイト未満であっても論理ページにまたがっている場合、全体の領域に合わせるために最初の論理ページの後に、十分な大きさの論理ページが残っていなくてはなりません。

アプリケーションがもし拡張メモリと内部メモリを交換する必要があれば、カレントのマッピングコンテキストを保存または復元せずに、対象の領域を交換することができます。交換のオペレーションが実行される前に、領域の重複がないかどうかチェックを行う必要があります。交換におけるソースとデスティネーションとの領域の重複は無効であり、交換は実行されません。

また、各ストラクチャのメンバは次のようになります。

.region_length

最初のメンバはダブルワードで、交換されるメモリ領域の大きさ（バイト）を指定します。

.source_memory_type

2 番目のメンバはバイトで、ソース領域のメモリのタイプを指定します。これが 0 ならば、ソース領域は内部メモリ内（ページフレームセグメントを除く）に存在し、1 ならば拡張メモリ内に存在することを示します。

.source_handle

3 番目のメンバはワードで、ソース領域が拡張メモリ内にある場合にソースメモリ領域と関連する EMM ハンドル番号を指定します。また、ソース領域が内部メモリ内にある場合は、この変数は意味がなく将来の互換性のために 0 にセットしなければなりません。

.source_initial_offset

4 番目のメンバはワードで、交換を開始するソース領域内のオフセットを指定します。ソース領域が拡張メモリ内にある場合、source_initial_offset は 16K バイトの論理ページの最初のアドレスを基準にするので、オフセットは 0000H から 3FFFH までの値をとります。またソース領域が内部メモリ内にある場合、source_initial_offset は交換を開始するソースセグメントの最初のオフセットを指定します。このオフセットは 64K バイトの内部メモリの最初のアドレスを基準にしているため、この値は 0000H から FFFFH までの値になります。

.source_initial_seg_page

5 番目のメンバはバイトで、交換を開始するソース領域のセグメント、または論理ページ番号を指定します。ソース領域が拡張メモリ内にある場合、交換を開始するソース領域の論理ページを指定します。また、ソース領域が内部メモリ内にある場合、source_initial_seg_page は交換を開始する内部メモリの最初のセグメントアドレスを指定します。

.dest_memory_type
6番目のメンバはバイトで、デスティネーション領域が存在するメモリのタイプを指定します。0ならば内部メモリで、1ならば拡張メモリに存在することを示します。

.dest_handle
7番目のメンバはワードで、デスティネーション領域が拡張メモリ内にある場合、デスティネーションメモリ領域に関連するEMMハンドル番号を指定します。デスティネーション領域が内部メモリ内にある場合、この変数は意味がなく将来の互換性のために0にセットしなければなりません。

.dest_initial_offset
8番目のメンバはワードで、交換を開始するデスティネーション領域内のオフセットを指定します。デスティネーション領域が拡張メモリ内にある場合、dest_initial offsetは16Kバイトの論理ページの最初のアドレスを基準にして、オフセットは0000Hから3FFFHまでの値をとります。また、デスティネーション領域が内部メモリ内にある場合、dest_initial_offsetは交換を開始するデスティネーションセグメントの、最初のオフセットを指定します。このオフセットは64Kバイトの内部メモリの最初のアドレスを基準にしているため、この値は0000HからFFFFHまでの値をとります。

.dest_initial_seg_page
9番目のメンバーはワードで、交換を開始するデスティネーション領域のセグメントまたは論理ページ番号を指定します。デスティネーション領域が拡張メモリ内にある場合、交換を開始するデスティネーション領域の論理ページを指定します。また、デスティネーション領域が内部メモリ内にある場合、dest_initial_seg_pageは交換を開始する内部メモリの最初のセグメントアドレスを指定します。

サンプル

```
xchg_source_dest            xchg_source_dest_struct(?)

MOV   AX,SEG xchg_source_dest    ;set xchg source dest segment
MOV   DS,AX                      ;
LEA   SI,xchg_source_dest        ;DS：SI xchg source dest pointer
MOV   AX,5701H                   ;load function code
INT   67H                        ;call the memory manager
OR    AH,AH                      ;check EMM status
JNZ   EMM_ERR_HANDLER            ;jump to error handler on error
```

INT 67H

ファンクション 5800H マップ可能な物理アドレス配列の取得

コール

AX = 5800H
ES：DI = mappable_phys_page
メモリマネージャが物理アドレス配列をコピーするアプリケーションが、供給しているメモリエリアへのポインタ。

```
mappable_phys_page_struct      STRUC
  phys_page_segment            DW ?
  phys_page_number             DW ?
mappable_phys_page_struct      ENDS
```

リターン

AH = 00H **正常実行**（物理ページのセグメントアドレス配列が返された）
= 80H **回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H **回復不可**（拡張メモリハードウェアが動作不能）
= 83H **回復不可**（指定のEMMハンドルがない）
= 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH **回復不可**（ファンクションパラメータが無効）
CX = mappable_phys_page **内のエントリ数**
物理ページアドレス配列が要求しているバイト数を決めるためには、mappable_phys_page_structのサイズにこの数値をかけます。

解　説

システム内のマッピング可能な各物理ページに対する物理ページ番号と、セグメントアドレスを含む配列を返します。

この配列はシステム内のマッピング可能な物理ページ番号と、実際のセグメントアドレスとの間のクロスリファレンスを与えるものです。また、この配列ではセグメントが昇順に並んでいます。これはセグメントアドレスに対応する物理ページも、昇順に並んでいることを意味しているわけではありません。

また、各ストラクチャのメンバは次のようになります。

.phys_page_segment

1番目のメンバはワードで、後に続く物理ページ番号に対応するマッピング可能な物理ページのセグメントアドレスです。配列のエントリは、昇順に並んだセグメントアドレスです。

.phys_page_number

2番目のメンバはワードで、前のセグメントアドレスに対応する物理ページ番号です。なお、物理ページ番号は昇順に並んでいるとは限りません。

サンプル

```
mappable_phys_page              mappable_phys_page_struct(?)
mappable_page_entry_count       DW ?

MOV   AX,SEG mappable_phys_page ;set mappable phys page segment
MOV   ES,AX                     ;
LES   DI,mappable_phys_page     ;ES:DI mappable phys page pointer
MOV   AX,5800H                  ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
MOV   mappable_page_entry_count,CX
                                ;save mappable page entry count
```

INT 67H

ファンクション 5801H マップ可能な物理アドレス配列エントリの取得

コール　AX = 5801H

リターン

AX = 00H　**正常実行**（物理アドレス配列のエントリを返した）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8FH　**回復不可**（ファンクションパラメータが無効）

CX = mappable_phys_page **のエントリ数**
このエントリ数は、システム内のマッピング可能な物理ページ数をも示します。物理ページアドレス配列が必要としているバイト数を決めるためには、mappable_phys_page_structのサイズにこの数値をかけます。

解　説

マッピング可能な物理アドレス配列のエントリを取得します。

このファンクションは、最初のファンクション5800H（マップ可能な物理アドレス配列の取得）が返す配列に必要とされるエントリの数を返します。

サンプル

```
mappable_page_entry_count          DW ?

MOV    AX,5801H                     ;load function code
INT    67H                          ;call the memory manager
OR     AH,AH                        ;check EMM status
JNZ    EMM_ERR_HANDLER              ;jump to error handler on error
MOV    mappable_page_entry_count,CX ;save mappable page entry count
```

INT 67H

ハードウェア構成配列の取得

コール

AX　　= 5900H
ES : DI = hardware_info
　メモリマネージャが拡張メモリハードウェア情報をコピーするOSが、供給しているメモリへのポインタ。

```
hardware_info_struct          STRUC
  raw_page_size               DW ?
  alternate_register_sets     DW ?
  context_save_area_size      DW ?
  DMA_register_sets           DW ?
  DMA_channel_operation       DW ?
hardware_info_struct          ENDS
```

リターン

AH = 00H　**正常実行**（ハードウェア構成配列が返された）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 83H　**回復不可**（指定のEMMハンドルがない）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8FH　**回復不可**（ファンクションパラメータが無効）
　= A4H　**回復不可**（このファンクションにアクセスすることをOSが禁止している。この場合このファンクションは使用不可）
hard_ware_info　**拡張メモリハードウェアの情報**

解　説

このファンクションは、OS/E環境によって使用される拡張メモリのハードウェア構成情報を含む配列を返します。

また、各ストラクチャのメンバは次のようになります。

.raw_page_size

1番目のメンバはワードで、マッピング可能なローページのサイズをパラグラフ単位（16バイト）で示します。標準ページ（PC–9800シリーズでのローページ）は16Kバイトです。しかし、他の拡張メモリボードの動作ではこの標準サイズに従っているわけではなく、複数のより小さなページをマッピング

することによって、16Kバイトのページをエミュレートできます。このメンバはハードウェアの動作レベルから見た、マッピング可能なページのサイズを指定するものです。

.alternate_register_sets

2番目のメンバはワードで、変更するマップレジスタのセットの数が入ります。追加のマップレジスタセットを、このマニュアルでは代替マップレジスタセットと呼んでいます。すべての拡張メモリボードは論理ページから物理ページへのマッピングを実行するために、少なくとも1つのハードウェアレジスタのセットを持っています。また、拡張メモリボードの中には、1つ以上のマップレジスタを持っているものもあります。このメンバはシステム内の代替マップレジスタセットが、いくつ存在するかが入ります（すべての拡張メモリが持っている1セットは除く）。もし拡張メモリボードがマップレジスタのセットを1つしか持っていない場合（つまり代替マップレジスタのセットがない場合）、このメンバの値は0です。

.context_save_area_size

3番目のメンバはワードで、マッピングコンテキストをセーブするために必要な配列の大きさが入ります。このメンバに返される値は、ファンクション4E03H（ページマップセーブ配列のサイズ取得）で返される値とまったく同じです。

.DMA_register_sets

4番目のメンバはワードで、DMAチャネルに割り当てられるレジスタセットの数が入ります。このDMAレジスタセットは、代替レジスタセットの使用方法と似ていますが、DMAのマップ用でありタスクのマップ用ではありません。もし、拡張メモリハードウェアがDMAレジスタセットをサポートしていなければ、DMAを実行するときは注意しなければなりません。マルチタスクOSではあるタスクがDMAを完了するのを待っている場合、別のタスクにスイッチを切り換えるのに便利です。しかし、次のタスクがリマップするための必要なメモリをDMAが操作していた場合は、リマップの結果は保証されません。また、拡張メモリハードウェアがDMAの動作状態を知ることができれば、OS/EはDMAの間のタスク切り換えとリマッピングを許可するべきです。DMAの特別のサポートがなければ、DMA実行中にはリマップを行うべきではありません。

.DMA_channel_operation

5番目のメンバはワードで、DMAレジスタセット用の特別な場合が入ります。この値が0ならば、DMAレジスタセットは記述してあるものとして機能します。また、1ならば拡張メモリハードウェアはDMAレジスタセットを1つだけしか所有しません。さらに、どれかのチャネルがこのレジスタセットを通じてマッピングされた場合、全チャネルはこのレジスタを通じてマッピングされます。EMS標準ボードではこの値は0です。

注意　このファンクションはOSのみが使用できます。また、OSによっていつでも使用禁止にすることが可能です。その方法については、ファンクション5D00H（OS/Eファンクションセットの使用許可）を参照してください。

サンプル

```
hardware_info               hardware_info_struct(?)

MOV     AX,SEG_hardware_info        ;set hardware info segment
MOV     ES,AX                       ;
LEA     DI,hardware_info            ;ES:DI hardware info offset
MOV     AX,5900H                    ;load function code
INT     67H                         ;call the memory manager
OR      AH,AH                       ;check EMM status
JNZ     EMM_ERR_HANDLER             ;jump to error handler on error
```

INT 67H

未アロケートのローページ数の取得

コール　AX = 5901H

リターン　AX = 00H　**正常実行**（未アロケートローページ数とローページの総数を返した）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 83H　**回復不可**（指定の EMM ハンドルがない）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（ファンクションパラメータが無効）

BX =**未アロケートのローページ数**
現在使用可能なローページの数。

DX =**ローページの総数**
拡張メモリ内のローページの総数。

解　　説

OS に対する拡張メモリ内の（標準サイズではない）マッピング可能なページ総数と、アロケートされていない（標準サイズではない）マッピング可能なページ数を返します。

ある種類の拡張メモリボードは、16K バイトの約数となるようなページサイズを持つものがあり、16K バイトの約数となるような拡張メモリページはローページと呼ばれます。OS は 16K バイトの約数となるようなマッピング可能な物理ページを処理することもあります。

もし、拡張メモリボードがちょうど 16K バイトの倍数サイズのページを供給している場合は、このファンクションが返す数はファンクション 42H（未アロケートページ数の取得）が返す値と同じになります。

注意　PC-9800 シリーズでは、標準ページとローページは同じサイズ（16K バイト）です。

サンプル

```
unalloc_raw_pages           DW ?
total_raw_pages             DW ?

MOV   AX,5901H              ;load function code
INT   67H                   ;call the memory manager
```

```
        OR      AH,AH                       ;check EMM status
        JNZ     EMM_ERR_HANDLER             ;jump to error handler on error
        MOV     unalloc_raw_pages,BX        ;save unalloc raw pages
        MOV     total_raw_pages,DX          ;save total raw pages
```

INT 67H

標準サイズのページのアロケートと固有のEMM ハンドルの割り当て

コール

AX = 5A00H
BX = num_of_standard_pages_to_alloc
　OS がアロケートしようとする標準ページ数。

リターン

AH = 00　**正常実行**（メモリマネージャが割り当てられた EMM ハンドルにページをアロケートした）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 85H　**復旧可能**（すべての EMM ハンドルが使用されている）
　= 87H　**復旧可能**（システム内に OS の要求を満たす数の拡張メモリページがない）
　= 88H　**復旧可能**（システム内に OS の要求を満たす数の未アロケートページがない）
　= 8FH　**回復不可**（ファンクションパラメータが無効）
DX = EMM ハンドル
　固有の EMM ハンドル。OS においては、この EMM ハンドルをパラメータとして必要とするすべてのファンクションで、このハンドルを使用しなければななりません。255 個までの EMM ハンドルを使用できます（このファンクションとファンクション 43H（ページのアロケート）は、同じ 255 個のハンドルを共有しなければなりません）。この EMM ハンドルを使用するすべてのファンクションについて、これにアロケートされている物理および論理ページの長さは、標準サイズ（16K バイト）になります。

解　説

OS が要求する数だけ標準サイズ（16K バイト）のページをアロケートし、各ページに固有の EMM ハンドルを割り当てます。この EMM ハンドルは、OS が EMM ハンドルを解除するまでこれらのページを処理します。このファンクションは、ファンクション 43H（ページのアロケート）とは異なり、EMM ハンドルに 0 ページをアロケートすることができます。

注意　次の事項は、拡張メモリマネージャインプリメンタと OS 開発者のみに関係するものです。アプリケーションのユーザーは、このメモリマネージャの特質を利用することはできません。こ

れらの規則を守らない場合は、マイクロソフト社のOSと互換性がなくなります。

拡張メモリマネージャにはこの仕様に互換性をもたせるために、OSのみが使用可能なハンドルが準備されています。このハンドルの値は0000Hで、拡張メモリマネージャがインストールされたときに、このハンドルにアロケートされたページの1セットが与えられます。メモリマネージャが0000Hのハンドルに自動的にアロケートするページは、メモリの上位アドレスに存在し、一般的にこれは40000H（256K）から9FFFFH（640K）のアドレス間にあります。しかし、この範囲でハードウェアやメモリマネージャをサポートしている場合は、この範囲を上下に拡張することができます。拡張メモリデバイスドライバがインストールされると、このハンドルはすでに存在していると想定され、すぐに使用可能になります。したがって、このハンドルを得るために本ファンクションを呼び出す必要はありません。
OSがこのハンドルを使用する場合は、0000Hの特別なハンドル値を使用して、任意のEMMファンクションを呼び出すことができます。また、このハンドルにページをアロケートするためにはファンクション51H（ページの再アロケート）を呼び出します。
このハンドルには次のように2つの特別な場合があります。

1. ファンクション5A00H（標準サイズのページのアロケートと固有のEMMハンドルの割り当て）
   このファンクションはハンドル値として0を返すことはありません。アプリケーションにおいてはページをアロケートし、そのページをもつハンドルを得るためには、ファンクション5A00Hを呼び出さなければなりません。ファンクション5A00Hは0のハンドル値を返すことはないので、アプリケーションでこの特別なハンドルにアクセスすることはできません。

2. ファンクション45H（ページのデアロケート（開放））
   OSが特別なEMMハンドルにアロケートされたページの解除のために、このファンクションを使用すると、EMMハンドルが所有するページは使用可能になり、メモリマネージャに返されます。しかし、このEMMハンドルは再び割り当てすることはできません。メモリマネージャはこのファンクションのEMMハンドルへの要求を、ファンクション43H（ページのアロケート）の要求と同じものとして取り扱います。したがって、このEMMハンドルへのリアロケートのページ番号は0になります。

サンプル

```
num_of_standard_pages_to_alloc  DW ?
emm_handle                      DW ?

MOV   BX,num_of_standard_pages_to_alloc
MOV   AX,5A00H                          ;load function code
INT   67H                               ;call the memory manager
OR    AH,AH                             ;check EMM status
JNS   EMM_ERR_HANDLER                   ;jump to error handler on error
MOV   emm_handle,DX                     ;save handle
```

INT 67H

ローページのアロケートと固有の EMM ハンドルの割り当て

コール

AX ＝ 5A01H
BX ＝ num_of_raw_pages_to_alloc
　OS がアロケートしようとするローページ数。

リターン

AH ＝ 00H　**正常実行**（メモリマネージャが割り当てられた EMM ローハンドルにローページをアロケートした）
　＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　＝ 85H　**復旧可能**（すべての EMM ハンドルが使用されている）
　＝ 87H　**復旧可能**（システム内に OS の要求を満たす数の拡張メモリローページがない）
　＝ 88H　**復旧可能**（システム内に OS の要求を満たす数の未アロケートローページがない）
　＝ 8FH　**回復不可**（ファンクションパラメータが無効）

DX ＝ **EMM ローハンドル**
　固有の EMM ローハンドル。OS においては、この EMM ローハンドルをパラメータとして必要とするすべてのファンクションで、このハンドルを使用しなければなりません。また、255 個までの EMM ハンドルを使用できます（ファンクション 5A00H とファンクション 43H は、同じ 255 個の EMM ハンドルを共有しなければなりません）。この EMM ローハンドルを使用するすべてのファンクションについて、これにアロケートされている物理および論理ページの長さは、標準サイズ（16K バイト）ではありません。

解　説

OS が要求する数だけ、標準サイズ（16K バイト）ではないページ（ローページ：PC–9800 シリーズのローページのサイズは 16K バイト）をアロケートし、各ページに固有の EMM ハンドルを割り当てます。

この EMM ハンドルは OS が EMM ハンドルを解除するまで、これらのページを処理します。このファンクションはファンクション 43H（ページのアロケート）とは異なり、EMM ハンドルに 0 ページをアロケートすることができます。

しかしハードウェアによっては、16Kバイトの約数になるページサイズをもつ拡張メモリボードがあります。このページサイズのことを、ロー（raw）ページと呼んでいます。このローページはOSが処理する場合もあります。このような場合には、メモリマネージャはローページがアロケートされたハンドルを使用して、すべてのファンクションを取り扱わなければなりません。これは、このファンクションを使用して処理を行いますが、次の2つの方法で標準（16Kバイト）のページがアロケートされているEMMハンドルを用いる場合とは、異なる使い方をします。

・ファンクション43H（ページのアロケート）や、ファンクション5A00H（標準サイズのページのアロケートと固有のEMMハンドルの割り当て）を使用して割り当てられたEMMハンドルは、16Kバイトのページを持っていなければなりません。これが標準的な拡張メモリページの大きさです。もし、拡張メモリボードハードウェアが16Kバイトのページを用意できなければ、メモリマネージャは複数の標準サイズではないページを組み合わせて、1つが16Kバイトのページを形成することで、16Kバイトのページをエミュレートしなければなりません。

・ファンクション5A01H（ローページのアロケートと固有のEMMハンドルの割り当て）を使用して割り当てられたEMMハンドルは、ローハンドルと呼ばれます。ローハンドルにアロケートされる論理ページは、すべて標準的ではない大きさ（16Kではないサイズ）になります。しかし、いったんOSが、ハンドルに複数のローページをアロケートしてしまうと、そのローハンドルに標準サイズではないページがアロケートされていることをメモリマネージャが識別します。このようにメモリマネージャには、これらのハンドルを識別し、標準サイズではないページをもつハンドルを使用するすべてのファンクションを、別々に取り扱うようにする役割があります。論理ページサイズはファンクション5A01Hが割り当てるいずれかのEMMローハンドルについても、標準サイズではないページになります。

注意　次の事項は、拡張メモリマネージャインプリメンタとOS開発者のみに関係するものです。アプリケーションのユーザーは、このメモリマネージャの特質を利用することはできません。これらの規則を守らない場合は、マイクロソフト社のOSと互換性がなくなります。

拡張メモリマネージャにはこの仕様に互換性をもたせるために、OSのみが使用可能なハンドルが準備されています。このハンドルの値は0000Hで、拡張メモリマネージャがインストールされたときに、このハンドルにアロケートされたページの1セットが与えられます。メモリマネージャが0000Hのハンドルに自動的にアロケートするページはメモリの上位アドレスに存在し、一般的にこれは40000H（256K）から9FFFFH（640K）のアドレス間にあります。しかし、この範囲でハードウェアやメモリマネージャをサポートしている場合は、この範囲を上下に拡張することができます。拡張メモリデバイスドライバがインストールされると、このハンドルはすでに存在していると想定され、すぐに使用可能になります。したがって、このハンドルを得るために本ファンクションを呼び出す必要はありません。
OSがこのハンドルを使用する場合は、0000Hの特別なハンドル値を使用して、任意のEMMファンクションを呼び出すことができます。また、このハンドルにページをアロケートするためにはファンクション51H（ページの再アロケート）を呼び出します。
このハンドルには次のように2つの特別な場合があります。

1. ファンクション5A00H（標準サイズのページのアロケートと固有のEMMハンドルの割

り当て)
このファンクションはハンドル値として0を返すことはありません。アプリケーションにおいてはページをアロケートし、そのページをもつハンドルを得るためには、ファンクション5A00Hを呼び出さなければなりません。ファンクション5A00Hは0のハンドル値を返すことはないので、アプリケーションでこの特別なハンドルにアクセスすることはできません。

2. ファンクション45H(ページのデアロケート(開放))
OSが特別なEMMハンドルにアロケートされたページの解除のために、このファンクションを使用すると、EMMハンドルが所有するページは使用可能になりメモリマネージャに返されます。しかし、このEMMハンドルは再び割り当てすることはできません。メモリマネージャはこのファンクションのEMMハンドルへの要求を、ファンクション43H(ページのアロケート)の要求と同じものとして取り扱います。したがって、このEMMハンドルへのリアロケートのページ番号は0になります。

サンプル

```
num_of_raw_pages_to_alloc       DW  ?
emm_raw_handle                  DW  ?

MOV   BX,num_of_raw_pages_to_alloc
MOV   AX,5A01H                  ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNS   EMM_ERR_HANDLER           ;jump to error handler on error
MOV   emm_raw_handle,DX         ;save raw handle
```

INT 67H

マップレジスタの変更

このファンクションはOSのみが使用します。OSはいつでもこのファンクションを使用禁止にすることができます。OSがこのファンクションを使用可または使用不可にする方法についてはファンクション5D00H（OS/Eファンクションセットの使用許可）を参照してください。

参　考

●設計構想

記述されているファンクションをサポートしているハードウェアは、どんな拡張メモリボード上にもあるとは限りません。あるファンクションにはソフトウェアエミュレーションが用意されていて、また他のファンクションには、使用に際して一定のプロトコルが実行されなくてなりません。システムDMA機能を利用するファンクションは、特にこのことが重要になります。

●システムDMA機能とDMAによる拡張メモリサポート

マルチタスクOSでは、あるタスクがDMAの完了するのを待っている場合など、別なタスクに切り換えができれば便利です。この仕様はDMAが発生している間、マップアウトされているメモリ領域にDMAができるように、拡張メモリボードに設計されている機能について記述します。この機能を備えていない拡張メモリボードはマッピング可能なメモリ領域でDMAを実行中、マッピングコンテキストを変更することはできません。つまり、すべてのDMA動作はページが再マッピングされる前に完了しておかなくてはなりません。

●拡張メモリによるDMAレジスタセットのサポート

DMAレジスタセットを持つ拡張メモリボードは、マッピングコンテキストが切り換えられている間にマッピング可能な領域にDMAを行うことができます。

OSのDMAレジスタセットの使用例は次のとおりです。

1. DMAレジスタセットのアロケート
2. 現在のレジスタの取得
3. DMAレジスタセットの設定
4. 要求されたメモリ内へのマップ
5. DMAレジスタセットの取得
6. 本来のレジスタセットの設定
7. DMAレジスタセットへ希望するDMAチャネルの割り当て

前述のコールにより、要求されたDMAチャネルへ全DMAアクセスが行われる場合は、現在のレジスタセットに関係なく、現在のDMAレジスタセットを通じてマッピングできるようになります。言い換えればDMAレジスタセットは、

指定の DMA チャネル上の DMA オペレーションのために現在のマップレジスタを無視することになります。DMA レジスタセットに割り当てられていない DMA チャネルは、そのすべての DMA を現在のマップレジスタセットを通してマッピングされます。

INT 67H

代替マップレジスタセットの取得

コール　AX = 5B00H

リターン

AH = 00H **正常実行**（代替マップレジスタセットを取得した）
= 80H **回復不可**（メモリマネージャやソフトウェアが動作不能）
= 81H **回復不可**（拡張メモリハードウェアが動作不能）
= 83H **回復不可**（指定のEMMハンドルがない）
= 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH **回復不可**（ファンクションパラメータが無効）
= A4H **回復不可**（OSがこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

BL **=カレントのアクティブな代替レジスタマップセットの番号**

BL ≠ 0 ファンクションがコールされているときに使用されているマップレジスタセットを含むES：DIは使用されない。

BL = 0 システム内の全マップレジスタの状態を含むエリアへのポインタ。また本来の状態に戻すために返すに必要な追加情報が返されたことを示す。

ES：DI **=マップレジスタコンテキストのセーブエリアへのポインタ**

OSが供給しているコンテキストのセーブエリアへのポインタを含みます。このポインタはセグメント：オフセットの形式を持ち、もし拡張メモリハードウェアが代替マップレジスタセットを供給していなければ、このポインタは常に返されます。OSはファンクション5B01H（代替マップレジスタセットの設定）をコールするときは、必ずこのポインタを拡張メモリマネージャに渡します。OSが代替マップレジスタセットのコールする前にこのファンクションをコールした場合、ポインタ値を0にして返します。OSはセーブエリア用にスペースを確保しておかなくてはなりません。しかし有効な情報が格納される前に、OSはメモリマネージャにこのセーブエリアの内容を初期化するように要求しなければなりません。

OSは、ファンクション4E00H（ページマップの取得）によりアロケートされたセーブエリアを初期化します。初期化終了後、そのセーブエリアにはシステム内の全ボード上のマップレジスタの状態が全部格納されます。このセーブエリアには、OSが代替マップレジスタセットの設定をコールするときに、本来の状態に戻すために必要な追加情報も格納されています。

解　　説

代替マップレジスタセットの取得をします。

このファンクションがコールされているときにアクティブになっているマップレジスタによって、次の2つのうちいずれかを実行します。

・代替マップレジスタセットの設定が、代替マップレジスタセットに0（BL＝0）を返して終了していた場合

1. 代替マップレジスタセットの設定によって、EMM内に保存されたコンテキスト保持領域ポインタは、この呼び出しによって返されます。このポインタは常に代替マップレジスタセットを用意しないボードのために返されます。

2. 返されたコンテキスト保存領域ポインタが0でなければ、このファンクションはシステムの各拡張メモリボードのマップレジスタの内容を、このポインタが指定する保存領域にコピーします。この保存領域の書式は、ファンクション4E00H（ページマップの取得）によって返されるものと同じになります。これは代替マップレジスタセットを得ることをシミュレートする目的で行われます。メモリマネージャはこのマッピングコンテキストにスペースをアロケートしないことに注意してください。これはOSが行います。

3. 返されたコンテキスト保存領域ポインタが0であれば、このファンクションはシステムの各拡張メモリボードのマップレジスタの内容を、このポインタが指定する保存領域にコピーすることはありません。

4. コンテキスト保存領域ポインタは、先行するファンクション5B01H（代替マップレジスタセットの設定）をコールすることよって、初期設定されていなければなりません。EMM内に保存されているコンテキスト保存領域ポインタは、インストールの直後は0であることに注意してください。

5. コンテキスト保存領域は先行するファンクション4E00H（ページマップの取得）をコールすることによって、初期設定されなければなりません。

・ファンクション5B01H（代替マップレジスタセットの設定）のコールが、0より大きい（BL＞0）代替マップレジスタセットを返して終了した場合

このファンクションがコールされたときに、使用されている代替マップレジスタセットの番号が返されます。

サンプル

```
alt_map_reg_set                    DB ?
context_save_area_ptr_seg          DW ?
context_save_area_ptr_offset       DW ?

MOV    AX,5B00H                     ;load function code
INT    67H                          ;call the memory manager
OR     AH,AH                        ;check EMM status
JNZ    EMM_ERR_HANDLER              ;jump to error handler on error
MOV    alt_map_reg_set,BX           ;save alt map reg set
TEST   BL,BL                        ;
JNZ    no_ptr_returned              ;
MOV    context_save_area_ptr_seg,ES ;
MOV    context_save_area_ptr_offset,DI ;
no_ptr_returned:
```

INT 67H

代替マップレジスタセットの設定

コール

AX　= 5B01H

BL　**=新しい代替マップレジスタセットの番号**

アクティブな代替マップレジスタセットの番号。

BL ≠ 0　マップレジスタコンテキストのリストアエリアへのポインタは要求されない。また ES : DI の内容は影響を受けず無視される。ボードがレジスタをサポートしていれば BL 内の指定された代替マップレジスタセットはアクティブになる。

BL = 0　システム内の全ボードの全マップレジスタの状態を含むエリアへのポインタと、本来の状態に戻すために必要な追加情報が ES : DI に返されたことを示す。

ES : DI　**=マップレジスタコンテキストのリストアエリアへのポインタ**

OS が供給している、マップレジスタコンテキストのリストアエリアへのポインタを含みます。このポインタはセグメント：オフセットの形をとり、拡張メモリハードウェアが代替マップレジスタセットを供給していなければ常に返されます。

OS がこのファンクションをコールするときは、いつでもこのポインタをメモリマネージャに引き渡されなければなりません。またリストアエリアのためにあらかじめスペースを確保しなくてはなりません。さらにこのリストアエリアの内容は、有効な情報を格納する前にメモリマネージャが初期化するように、OS が要求を出さなくてはなりません。

OS はファンクション 4E00H（ページマップの取得）によって確保したリストアエリアを初期化します。初期化終了後、このエリアはマップレジスタの状態を含むことになります。OS がファンクション 5B01H（代替マップレジスタセットのセット）をコールするとき、このリストアエリアは本来の状態に戻すために必要な追加情報も含みます。

リターン

AH = 00H　**正常実行**（代替マップレジスタを設定した）

= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）

= 81H　**回復不可**（拡張メモリハードウェアが動作不能）

= 83H　**回復不可**（指定の EMM ハンドルがない）

= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）

= 8FH　**回復不可**（ファンクションパラメータが無効）

= 9AH　**回復不可**（代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットがサポートされない）

= 9CH　**回復不可**（代替マップレジスタセットはサポートされているが、

指定の代替マップレジスタセットは0ではない)
= 9DH **回復不可**(代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットが定義されていないか、アロケートされていない)
= A3H **回復不可**(ソース配列の内容が違うか、ファンクションから渡されたポインタが無効)
= A4H **回復不可**(OSがこのファンクションのアクセスを拒否している。このときファンクションは使用不可)

解 説

代替マップレジスタセットの設定を行います。
このファンクションは、指定のマップレジスタセットに従って、次の2つのうち1つを実行します。

1. もし設定された代替マップレジスタセットが0であれば、そのマップレジスタセットがアクティブになります。マップレジスタコンテキスト復元領域ポインタが0でなければ、ES:DIによって示された復元領域の内容を、システム内の各拡張メモリボードのレジスタセットにコピーします。このポインタが0であれば、その内容はコピーされません。
   マップレジスタコンテキスト復元領域ポインタはその値にかかわらず、メモリマネージャ内に保存されます。この値はファンクション5B00H(代替マップレジスタセットの取得)によって使用されます。
   OSはこのコンテキスト復元領域ポインタを用意しなければなりません。このファンクションは、代替マップレジスタセットの設定をシミュレートする目的で使われます。メモリマネージャはコンテキスト保存領域ポインタを内部的に保存します。このマッピングコンテキストにスペースをアロケートするのはOSが行うことであることに注意してください。

2. もし指定の変更マップレジスタセットが0でなければ、指定のマップレジスタセットがアクティブです。OSがポイントとしているリストアエリアは使用されることはありません。

サンプル

```
alt_map_reg_set                     DB ?
context_restore_area_ptr_seg        DW ?
context_restore_area_ptr_offset     DW ?

MOV   AX,5B01H                       ;load function code
MOV   BL,alt_map_reg_set             ;set alt map reg set
TEST  BL,BL                          ;
JZ    no_ptr_passed                  ;
MOV   ES,context_restore_area_ptr_seg      ;
MOV   DI,context_restore_area_ptr_offset   ;
```

```
no_ptr_passed:
INT    67H                        ;call the memory manager
OR     AH,AH                      ;check EMM status
JNZ    EMM_ERR_HANDLER            ;jump to error handler on error
```

INT 67H

ファンクション
5B02H 代替マップセーブ配列のサイズ取得

コール

AX ＝5B02H

リターン

AH ＝00H　**正常実行**（配列の大きさを返した）
　＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　＝81H　**回復不可**（拡張メモリハードウェアが動作不能）
　＝83H　**回復不可**（指定のEMMハンドルがない）
　＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　＝8FH　**回復不可**（パラメータが無効）
　＝A4H　**回復不可**（OSがこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

DX ＝**配列のサイズ**
OSがファンクション5B00H（代替マップレジスタセットの取得）、5B01H（代替マップレジスタセットの設定）に要求するときに必ずOSが供給している、メモリエリアに転送されるバイト数を含みます。

解　説

代替マップレジスタコンテキストをセーブするために必要なサイズを返します。

サンプル

```
size_of_array                     DW ?

MOV   AX,5B02H                    ;load function code
INT   67H                         ;call the memory manager
OR    AH,AH                       ;check EMM status
JNZ   EMM_ERR_HANDLER             ;jump to error handler on error
MOV   size_of_array,DX            ;save size of array
```

INT 67H

代替マップレジスタセットのアロケート

コール　AX = 5B03H

リターン

AH = 00H　**正常実行**（代替マップレジスタの番号を返した）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 83H　**回復不可**（指定の EMM ハンドルがない）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8FH　**回復不可**（パラメータが無効）
　= 9BH　**回復不可**（代替マップレジスタはサポートされているが、現在代替マップレジスタはすべてアロケートされている）
　= A4H　**回復不可**（OS がこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

BL =**代替マップレジスタセットの番号**

代替マップレジスタセットの番号を含みます。もしハードウェアが代替マップレジスタセットをサポートしていなければ、0 が返されます。この場合マップレジスタコンテキストセーブエリアへのポインタを取得するために、ファンクション 5B00H（代替マップレジスタセットの取得）がコールされなければなりません。OS は、このセーブエリアをサポートする必要があります。このセーブエリアは、ハードウェアが変更マップレジスタセットをサポートしていないため必要になります。

解　　説

使用できる代替マップレジスタセットの番号を返します。もしハードウェアが代替マップレジスタをサポートしていなければ、代替マップレジスタセットの番号には 0 が返されます。

アロケートされた代替マップレジスタセットは、ファンクション 5B00H（代替マップレジスタセットの取得)、5B01H（代替マップレジスタセットの設定）を使用するときに、この返された番号で参照されます。OS はこれらのファンクションを使って代替マップレジスタを参照し、マッピングコンテキストを拡張メモリ上ですぐ切り換えることができます。

またこのファンクションは、アクティブな代替マップレジスタセットの内容を、新しくアロケートされた代替マップレジスタセットのマップレジスタにコピーします。これにより、OS がファンクション 5B01H（代替マップレジスタセットの設定）を実行するときに、新しい代替マップのアロケートの前にマッピングされたメモリへの読み書きが可能になります。代替マップレジスタセットを、実際に使用中

のマップレジスタセットを変更することはありませんが、新しい代替マップレジスタセットをアロケートすることで、後に続くファンクション5B01H（代替マップレジスタセットのセット）のために新しい代替マップレジスタセットを用意します。

サンプル

```
alt_map_reg_unm                        DB ?

MOV    AX,5B03H                       ;load function code
INT    67H                            ;call the memory manager
OR     AH,AH                          ;check EMM status
JNZ    EMM_ERR_HANDLER                ;jump to error handler on error
MOV    alt_map_reg_num,BL             ;save alt map reg num
```

INT 67H

ファンクション
5B04H 代替マップレジスタセットの開放

コール

AX ＝ 5B04H
BL ＝**代替マップレジスタセットの番号**
開放される代替マップレジスタセットの番号。マップレジスタセットが 0 ならばアロケートまたは開放はできません。ただし、変更マップレジスタセットに 0 が指定され、このファンクションがコールされていた場合は、エラーにならずにこのコールは無視されます。

リターン

AH ＝ 00H　**正常実行**（代替マップレジスタセットは開放された）
＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H　**回復不可**（指定の EMM ハンドルがない）
＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8FH　**回復不可**（パラメータが無効）
＝ 9CH　**回復不可**（代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットが 0 でない）
＝ 9DH　**回復不可**（代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットが定義されていないか、アロケートされていない）
＝ A4H　**回復不可**（OS がこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

解　説

代替マップレジスタセットを、将来の使用のためにメモリマネージャに返します。メモリマネージャは必要なときにこの代替マップレジスタを再アロケートできます。

またこのファンクションは、指定の代替マップレジスタのマッピングコンテキストを読み書き不可能（解除）にもできます。これにより、以前にこの代替マップレジスタにマッピングされたページをアクセス不可にして保護します。このとき現在の代替マップレジスタセットは開放はできません。また、これを実行すると現在の内部メモリと拡張メモリにマッピングされているすべてのメモリが、アクセス不可になります。

サンプル

```
alternate_map_reg_set           DB ?

MOV   BX,alternate_map_reg_set  ;set alternate map reg set
MOV   AX,5B04H                  ;load function code
INT   67H                       ;call the memory manager
OR    AH,AH                     ;check EMM status
JNZ   EMM_ERR_HANDLER           ;jump to error handler on error
```

INT 67H

ファンクション 5B05H　DMA レジスタセットのアロケート

コール　AX = 5B05H

リターン　AH = 00H　**正常実行**（DMA レジスタセットをアロケートした）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 83H　**回復不可**（指定の EMM ハンドルがない）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（パラメータが無効）
= 9BH　**回復不可**（DMA レジスタセットはサポートされているが、現在 DMA レジスタセットは全部アロケートされている）
= A4H　**回復不可**（OS がこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

BL = **DMA レジスタセットの番号**
DMA レジスタセットの番号を含みます。もし DMA レジスタセットがハードウェアによりサポートされていない場合は 0 が返ります。

解　説

DMA レジスタセットが現在使用可能ならば、DMA レジスタセットの番号を返します。もしハードウェアが DMA レジスタセットをサポートしていなければ、DMA レジスタセット番号には 0 が返されます。

マルチタスク OS では、あるタスクが DMA の完了するのを待っている場合など、別なタスクに切り換えができれば便利です。しかし、DMA が現在のレジスタセットを通じてマッピングされている間はこの切り換えはできません。つまり、すべての DMA 動作はページが再マッピングされる前に完了しておかなくてはなりません。

OS は指定の代替マップレジスタセットを使用して、指定の DMA チャネルでの DMA オペレーションを開始します。この代替マップレジスタセットは DMA オペレーションが完了するまで、OS またはアプリケーションにより再度使用されることはありません。OS は代替マップレジスタの内容を変更しないことと、DMA オペレーションの間アプリケーションが代替マップレジスタセットの内容を変更させないようにすることにより、DMA の動作を保証します。

注意　PC-9800 シリーズでは、DMA レジスタセットはサポートしていません。

サンプル

```
DMA_reg_set_number          DB ?

MOV   AX,5B04H              ;load function code
INT   67H                   ;call the memory manager
OR    AH,AH                 ;check EMM status
JNZ   EMM_ERR_HANDLER       ;jump to error handler on error
MOV   DMA_reg_set_number,BL ;save DMA reg set number
```

INT 67H

代替マップレジスタ上の DMA の使用許可

コール

AX ＝ 5B06H

BL ＝ **DMA レジスタセット番号**

DL で指定される DMA チャネル上の、DMA オペレーションに使用される代替マップレジスタセットの番号。もし指定の代替マップレジスタセットが 0 ならば、指定の DMA チャネルに対する DMA アクセスは実行されません。

DL ＝ **DMA チャネル番号**

BL で指定された DMA マップレジスタセットに対応する DMA チャネル

リターン

AH ＝ 00H　**正常実行**（代替マップレジスタの DMA の使用を許可した）
＝ 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H　**回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H　**回復不可**（指定の EMM ハンドルがない）
＝ 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8FH　**回復不可**（パラメータが無効）
＝ 9AH　**回復不可**（DMA レジスタセットはサポートされているが、指定の代替マップレジスタセットはサポートされていない）
＝ 9CH　**回復不可**（DMA レジスタセットはサポートされていない。また指定の DMA レジスタセットは 0 でない）
＝ 9DH　**回復不可**（DMA レジスタセットはサポートされているが指定の DMA レジスタセットは定義されていないか、アロケートされていない）
＝ 9EH　**回復不可**（DMA チャネルはサポートされていない）
＝ 9FH　**回復不可**（DMA チャネルはサポートされているが、指定の DMA チャネルはサポートされていない）
＝ A4H　**回復不可**（OS がこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

解　説

指定の代替マップレジスタセットを通じて、指定の DMA チャネルでの DMA アクセスを可能にします。マルチタスク OS では、あるタスクが DMA の完了するのを待っている場合など、別なタスクに切り換えができれば便利です。指定のチャネルでの DMA は、現在のレジスタセットと関係なく指定の

DMAレジスタセットを利用できます。もし、DMAチャネルがDMAレジスタセットに割り付けられていなければ、そのチャネルに対するDMAは現在のレジスタセットを通じてマッピングされます。

注意　PC–9800シリーズではDMAレジスタセットをサポートしていません。

サンプル

```
alt_map_reg_set                DB ?
DMA_channel_num                DB ?

MOV   BL,alt_map_reg_set       ;set alt map reg set
MOV   DL,DMA_channel_num       ;set DMA channel num
MOV   AX,5B06H                 ;load function code
INT   67H                      ;call the memory manager
OR    AH,AH                    ;check EMM status
JNZ   EMM_ERR_HANDLER          ;jump to error handler on error
```

INT 67H

ファンクション
5B07H 代替マップレジスタ上の DMA の使用不許可

コール

AX ＝ 5B07H
BL ＝**代替マップレジスタセットの番号**
オペレーションを無効にする DMA レジスタセットの番号。もし指定のマップレジスタセットが 0 ならば、何も処理されません。

リターン

AH ＝ 00H **正常実行**（代替マップレジスタの DMA を使用不可にした）
＝ 80H **回復不可**（メモリマネージャソフトウェアが動作不能）
＝ 81H **回復不可**（拡張メモリハードウェアが動作不能）
＝ 83H **回復不可**（指定の EMM ハンドルがない）
＝ 84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
＝ 8FH **回復不可**（パラメータが無効）
＝ 9AH **回復不可**（DMA レジスタセットはサポートされているが、指定の代替マップレジスタセットはサポートされていない）
＝ 9CH **回復不可**（DMA レジスタセットはサポートされていない。また指定の DMA レジスタセットは 0 でない）
＝ 9DH **回復不可**（DMA レジスタセットはサポートされているが指定の DMA レジスタセットは定義されていないか、アロケートされていない）
＝ 9EH **回復不可**（DMA チャネルはサポートされていない）
＝ 9FH **回復不可**（DMA チャネルはサポートされているが、指定の DMA チャネルはサポートされていない）
＝ A4H **回復不可**（OS がこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

解　説

指定の代替マップレジスタセットに対応する DMA チャネルへの DMA アクセスを不可能にします。

注意　PC-9800 シリーズでは DMA レジスタセットをサポートしていません。

サンプル

```
DMA_reg_set                     DB ?

MOV     BL,DMA_reg_set          ;set DMA reg set
MOV     AX,5B07H                ;load function code
INT     67H                     ;call the memory manager
OR      AH,AH                   ;check EMM status
JNZ     EMM_ERR_HANDLER         ;jump to error handler on error
```

INT 67H

DMA レジスタセットの開放

コール　AX = 5B08H

リターン　AH = 00H　**正常実行**（DMA レジスタセットは開放された）
= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
= 83H　**回復不可**（指定の EMM ハンドルがない）
= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
= 8FH　**回復不可**（パラメータが無効）
= 9CH　**回復不可**（DMA レジスタセットはサポートされていない。また指定の DMA レジスタセットは 0 でない）
= 9DH　**回復不可**（DMA レジスタセットはサポートされているが指定の DMA レジスタセットは定義されていないか、アロケートされていない）
= A4H　**回復不可**（OS がこのファンクションのアクセスを拒否している。このときファンクションは使用不可）

BL = **DMA レジスタセットの番号**
DMA オペレーションで使用しなくなる DMA レジスタセットの番号。これは以前に指定の DMA チャンネルでの DMA のためにアロケートされ、使用を許可されていたレジスタセットについて行います。もし指定の DMA レジスタセットが 0 ならば、何も処理されません。

解　説

指定の DMA レジスタセットを開放します。

注意　PC-9800 シリーズでは DMA レジスタセットをサポートしていません。

サンプル

```
DMA_reg_set_num                     DB ?

MOV     AX,5B08H                    ;load function code
INT     67H                         ;call the memory manager
OR      AH,AH                       ;check EMM status
JNZ     EMM_ERR_HANDLER             ;jump to error handler on error
MOV     DMA_reg_set_num,BL          ;save DMA reg set num
```

INT 67H

ファンクション 5CH　ウォームブートのための拡張メモリの準備

コール　AH ＝ 5CH

リターン

AH ＝ 00H	正常実行	（ハードウェアの準備ができた）
＝ 80H	回復不可	（メモリマネージャソフトウェアが動作不能）
＝ 81H	回復不可	（拡張メモリハードウェアが動作不能）
＝ 83H	回復不可	（指定のEMMハンドルがない）
＝ 84H	回復不可	（メモリマネージャに渡されたファンクションが未定義）

解　説

緊急のウォームブートのために拡張メモリハードウェアを準備します。このファンクションは、OSが次にする動作はシステムのウォームブートであるとみなします。一般にファンクションは、現在のマッピングコンテキストや使用中の代替マップレジスタセット、またウォームブート時に初期化の必要がある拡張メモリハードウェアに依存しているものすべてに影響します。もし、アプリケーションが640Kバイト以下のアドレスにメモリをマッピングしようとすれば、アプリケーションはウォームブートに続くすべての可能な状況をトラップし、アプリケーション自体をウォームブートする前にこのファンクションをコールしなくてはなりません。

注意　PC-9800シリーズでは常にAHが返されます。

サンプル

```
MOV     AH,5CH              ;load function code
INT     67H                 ;call the memory manager
OR      AH,AH               ;check EMM status
JNZ     EMM_ERR_HANDLER     ;jump to error handler on error
```

INT 67H

OS/E ファンクションセットの使用許可

コール

AX　　= 5D00H
BX、CX=アクセスキー
　2回目以降の全ファンクションコールで必要になります。アクセスキーは、最初のファンクションコールで返された値が要求されます。

リターン

AH = 00H　**正常実行**（OS ファンクションが使用可能になった）
　= 80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
　= 81H　**回復不可**（拡張メモリハードウェアが動作不能）
　= 84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）
　= 8FH　**回復不可**（パラメータが無効）
　= A4H　**回復不可**（OS が、このファンクションのアクセスを拒否している。このときファンクションは使用不可。このファンクションにより渡されるキーの値はこのファンクションの実行を許可するものではない）
BX、CX=アクセスキー
　最初のファンクションコール時に返されます。メモリマネージャは、以降このファンクションを実行するために必要なランダム値のキーを返します。2回目以降のファンクションコールではこのキーは返されません。このファンクションが2回目以降にコールされた場合は、BX、CX は影響を受けません。

解　説

OS 指定のファンクションを使用できるように、OS が全プログラムまたはデバイスドライバに使用許可を与えます。この機能は、マッピング可能な内部メモリの領域を管理する OS のためにのみ用意されており、プログラムがマッピング可能な内部メモリ領域に影響を与えるようなファンクションを使用することは許可されていません。しかし、この機能のためにはこのファンクションを使用することはできます。

OS がこのファンクションを使用禁止にしているとき、プログラムがこのファンクションを使用しようとすると、メモリマネージャは OS がファンクションへのアクセスを拒否しているというステータスを返します。これは使用禁止状態のときファンクションは動作しないということです。しかし使用許可状態のときはすべてのプログラムはファンクションを使用することが可能です。

ファンクションにより使用可能となる、OS/E ファンクションには次のものがあります。

1. ファンクション 5900H　　　　　　：ハードウェア構成配列の取得
2. ファンクション 5B01H　　　　　　：代替マップレジスタセットの設定
3. ファンクション 5D00H、5D01H：OS/Eファンクションセットの使用許可／不許可

これらのファンクションを再度使用可能にするファンクション自体が使用を禁止されている場合は、メモリマネージャがロードされたときに、このファンクションも含めてOS指定の全ファンクションを使用可にします。OSは他のソフトウェアがコールする前にファンクション 5D00H、5D01H（OS/Eファンクションセットの使用許可／不許可）をコールすることにより、これらのファンクションへの限られたアクセスを取得します。

このファンクションのうちどれか1つを最初にコールしたとき、メモリマネージャはOSがこれらのファンクションのどれかを、将来コールするときに使用しなければならないアクセスキーを返します。ただしメモリマネージャがファンクション 5D00H、5D01H（OS/Eファンクションセットの使用許可／不許可）を最初にコールするときには、アクセスキーは必要ありません。

以降、ファンクションをコールするときにはこのアクセスキーはファンクション 5D00H、5D01H（OS/Eファンクションセットの使用許可／不許可）で必要となります。アクセスキーは、これらのファンクションの最初のコールにのみ返されます。また、OSはこのファンクションをコールする最初のソフトウェアなので、OSのみがこのキーのコピーを得ることができます。メモリマネージャはランダム値でアクセスキーを返さなくてはなりません。固定値のキーを返すとOSへの安全保護の目的が失われます。

注意　このファンクションはOSによってのみ使用されます。OSはいつでもこのファンクションを使用禁止にできます。

サンプル

・最初のコール

```
access_key                          DW 2 DUP(?)

MOV    AX,5D00H                     ;load function code
INT    67H                          ;call the memory manager
OR     AH,AH                        ;check EMM status
JNZ    EMM_ERR_HANDLER              ;jump to error handler on error
MOV    access_key[0],BX             ;save access key
MOV    access_key[2],CX             ;
```

・2回目以降のコール

```
access_key                          DW 2 DUP(?)

MOV    BX,access_key[0]             ;set access key
MOV    CX,access_key[2]             ;
MOV    AX,5D00H                     ;load function code
INT    67H                          ;call the memory manager
OR     AH,AH                        ;check EMM status
JNZ    EMM_ERR_HANDLER              ;jump to error handler on error
```

INT 67H

ファンクション 5D01H OS/E ファンクションセットの使用不許可

コール

AX　　＝5D01H

BX、CX＝アクセスキー

2回目以降の、全ファンクションコールで必要になります。アクセスキーは、最初のファンクションコールで返された値が要求されます。

リターン

AH＝00H　**正常実行**（OS/E ファンクションを使用禁止にした）

＝80H　**回復不可**（メモリマネージャやソフトウェアが動作不能）

＝81H　**回復不可**（拡張メモリハードウェアが動作不能）

＝84H　**回復不可**（メモリマネージャに渡されたファンクションが未定義）

＝8FH　**回復不可**（パラメータが無効）

＝A4H　**回復不可**（OS が、このファンクションのアクセスを拒否している。このときファンクションは使用不可。このファンクションにより渡されたキーの値はこのファンクションを実行を許可するものではない）

BX、CX＝アクセスキー

最初のファンクションコール時のみ返されます。メモリマネージャは、以降このファンクションを実行するために必要なランダム値のキーを返します。2回目以降のファンクションコールでは、このキーは返されません。このファンクションが2回目以降にコールされた場合は、BX、CX は影響を受けません。

解　説

OS が全プログラムまたはデバイスドライバについて OS 指定のファンクションを使用できないようにします。この機能はマッピング可能な内部メモリの領域を管理する OS のためにのみ用意され、プログラムがマッピング可能な内部メモリ領域に影響を与えるようなファンクションを使用することはできません。この機能はこれらのファンクションを使用できなくてはなりません。OS がこのファンクションを使用禁止にしているとき、プログラムがこのファンクションを使用しようとすると、メモリマネージャは OS がファンクションへのアクセスを否定しているというステータスを返します。したがって使用状態のときは、ファンクションは動作しないということになります。

OS/E ファンクションで動作が可能なファンクションには次のものがあります。

1. ファンクション 5900H　　　　　：ハードウェア構成配列の取得
2. ファンクション 5B01H　　　　　：代替マップレジスタセットの設定
3. ファンクション 5D00H、5D01H：OS/E ファンクションセットの使用許可／不許可

注意　このファンクションはOSによってのみ使用されます。OSはいつでもこのファンクションを使用禁止にできます。

サンプル

・最初のコール

```
access_key                  DW 2 DUP(?)

MOV   AX,5D01H              ;load function code
INT   67H                   ;call the memory manager
OR    AH,AH                 ;check EMM status
JNZ   EMM_ERR_HANDLER       ;jump to error handler on error
MOV   access_key[0],BX      ;save access key
MOV   access_key[2],CX      ;
```

・2回目以降のコール

```
access_key                  DW 2 DUP(?)

MOV   BX,access_key[0]      ;set access key
MOV   CX,access_key[2]      ;
MOV   AX,5D01H              ;load function code
INT   67H                   ;call the memory manager
OR    AH,AH                 ;check EMM status
JNZ   EMM_ERR_HANDLER       ;jump to error handler on error
```

INT 67H

ファンクション 5D02H アクセスキーのリターン

コール

AX =5D02H
BX、CX=アクセスキー
　2回目以降の、全ファンクションコールで必要になります。アクセスキーは、最初のファンクションコールで返された値が要求されます。

リターン

AH =00H **正常実行**（アクセスキーがメモリマネージャに返された）
=80H **回復不可**（メモリマネージャソフトウェアが動作不能）
=81H **回復不可**（拡張メモリハードウェアが動作不能）
=84H **回復不可**（メモリマネージャに渡されたファンクションが未定義）
=8FH **回復不可**（パラメータが無効）
=A4H **回復不可**（OSが、このファンクションのアクセスを拒否している。このときファンクションは使用不可。このファンクションにより渡されたキーの値はこのファンクションの実行を許可するものではない）

解　説

OSがメモリマネージャにアクセスキーを返せるようにする機能です。メモリマネージャにアクセスキーを返すと、メモリマネージャはインストール時の状態（OS/Eファンクションセットとアクセスキーの使用に関して）になります。つまりOS/Eファンクションセットへのアクセスが使用許可になるわけです。次回のファンクション5D00H、5D01H（OS/Eファンクションセットの使用許可／不許可）のファンクションの実行により、アクセスキーが再び与えられます。

注意　このファンクションはOSによってのみ使用されます。OSはいつでもこのファンクションを使用禁止にすることができます。

サンプル

```
access_key                          DW 2 DUP(?)

MOV   BX,access_key[0]              ;set access key
MOV   CX,access_key[2]              ;
MOV   AX,5D02H                      ;load function code
INT   67H                           ;call the memory manager
OR    AH,AH                         ;check EMM status
JNZ   EMM_ERR_HANDLER               ;jump to error handler on error
```

INT 67H

ページフレームの管理

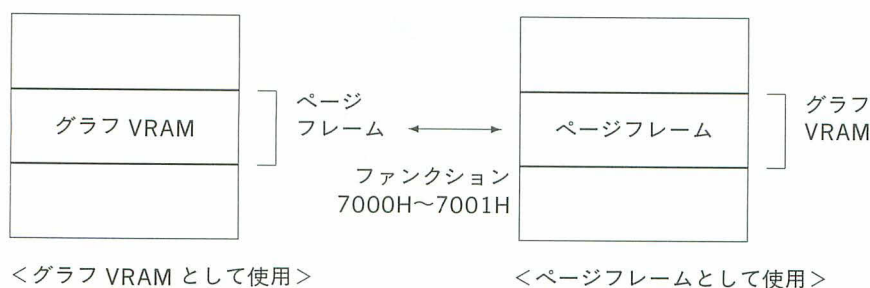
PC–9800シリーズではページフレームとして使用するメモリアドレスが、グラフィックVRAMの一部（B0000H〜BFFFFH）と重なっており、切り換えによってどちらでも使用できる機種（*）があります。

このような機種では、そのメモリとページフレームとして使用する場合、グラフィックVRAMとして使用する場合でアプリケーションがメモリを切り換えなければなりません。

(*) PC–9801RA、RX、LS、ES、EX、RS、LX、RL、DA、DS、DX、T、NS、NS/E、PC–H98（ノーマルモード）（EMS機能のある機種）

その他の機種（ノーマルモード）では、ページフレームはC000HまたはC8000H〜CFFFFHとなっています。ハイレゾリューションモードの機種では、ページフレームはB0000H〜BFFFFHとなっていますが、グラフィックVRAMはC000H〜DFFFFHとなっています。

これらの機種でこのファンクションを使用することができます。

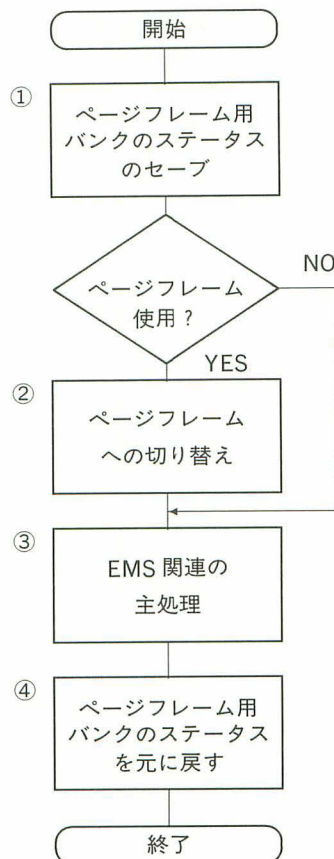
このファンクションは、グラフィックVRAMとページフレームの切り換え（または状態取得）のために提供されます。

このファンクションには、次のようなファンクションがあります。

1. ファンクション7000H：ページフレーム用バンクのステータスの取得
2. ファンクション7001H：ページフレーム用バンクの状態の設定

EMSドライバは次に示すファンクションを実行する場合はページフレーム用バンクに切り換えてしまうため、グラフィックVRAMを使用するアプリケーションは必要に応じてVRAMバンクに切り替えてください。

44H	ページのマップ
47H	ページマップのセーブ
48H	ページマップのリストア
4E00H、4E01H	ページマップのセーブ／リストア
4F00H、4F01H	ページマップの一部のセーブ／リストア
5000H	複数ページのマップ
55H	ページの変更とジャンプ
56H	ページマップの変更とコール
5700H	メモリ領域の移動
5B00H、5B01H	代替マップレジスタによるページマップの取得／設定

●ページフレーム用バンクの切り換え方法

ページフレーム用バンクがグラフィックVRAMの一部と重なっている機種において、EMSを使用するアプリケーションはページフレーム用バンクの切り換え操作を、次のような手順で行わなければなりません。

① アプリケーションの起動時にページフレーム用バンクのステータスを取得（ファンクション7000H）して、セーブしておいてください。

② ページフレーム用バンクをページフレームとして使用する場合、ページフレーム用バンクをページフレームに切り替えてください（ファンクション7001H）。ただし、ページフレーム用バンクがすでにページフレームに切り換わっている場合は、その必要はありません。

③ EMS関連の主な処理を行います（ページのアロケート、ページのマップ、ページフレームのアクセス等）。

④ アプリケーション終了時に、ページフレーム用バンクのステータスを①でセーブしていた内容に戻してください（ファンクション7001H）。

INT 67H

ファンクション
7000H ページフレーム用バンクのステータスの取得

コール　AX ＝7000H

リターン　AH ＝00H　**正常実行**（ステータスを取得した）
＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
AL ＝ステータス
00H　ページフレーム用バンクはページフレームとして使用可能
01H　ページフレーム用バンクはページフレームとして使用不可

解　説

ページフレーム用バンク（ページフレーム）の状態を返します。返された値によって現在ページフレーム用バンクのメモリを、ページフレームとして使用できるかどうかを判別します。

INT 67H

ファンクション 7001H ページフレーム用バンクの状態の設定

コール

AX ＝7001H
BL ＝指示
　00H　ページフレーム用バンクをページフレームとして使用可能にする
　01H　ページフレーム用バンクをVRAMとして使用する

リターン

AH ＝00H　**正常実行**（ファンクションは正しく実行された）
　＝80H　**回復不可**（メモリマネージャソフトウェアが動作不能）
AL ＝ステータス
　00H　ページフレーム用バンクはページフレームとして使用可能
　01H　ページフレーム用バンクはページフレームとして使用不可

解　説

指定された状態にページフレーム用バンクの切り換えを行います。OS、アプリケーションの開発者は、ファンクション7000H、7001Hの2つのファンクションでページフレーム（ページフレーム用バンク）を管理しなければなりません。

5.6 拡張メモリマネージャのインプリメンテーションへのガイドライン

ファンクションの機能仕様に加えて、拡張メモリマネージャはある資源を用意しています。EMS のこのバージョンにおいて、拡張メモリマネージャに必要とされる資源は次のようになります。

サポートされる拡張メモリの量

拡張メモリの量は、最大 32M バイトまでです。

サポートされるハンドル数

拡張メモリのハンドル数は最大数は 255 で、最小数は 64 です。

ハンドルの番号づけ

ハンドルは 1 ワードの大きさですが、OS のハンドル数も含めて、最大 255 個あります。この EMS では、次のようにハンドルワードを定義しています。ワードはメモリマネージャによって、下位バイトは実ハンドル値、上位バイトは 00 にセットされています。ただし、この EMS の前のバージョン（3.2 以前）では、ハンドルの値を指定していません。

新しいハンドルの型（ハンドルとローハンドル）

ロー（raw）ハンドルと標準的なハンドルの違いはほとんどありません。ファンクション 5A01H を使用すると、DX で返されるのはローハンドルになります。ローハンドルは必ずしも 16K バイトとは限りません。〝ロー（raw）〟ページの大きさに関係しないかわりに、それは拡張メモリハードウェアに依存するローページサイズとなります。

アプリケーションは、ファンクション 5901H によって、ローページの大きさを調べることができます。また、ファンクション 5A01H が返すローハンドルを使用することによって、特別な拡張メモリボードだけが許容する、より優れた方法でアクセスすることができます。

一方、ファンクション 43H を使用するアプリケーションは、必ず 16K バイトを扱うようなハンドルを割り当てられます。より小さなローページを持つ拡張メモリボートに関しては、EMM デバイスドライバが、ひとつの合成した 16K バイトのページを作るために必要とする、ローページ数を割り当てて保持します。拡張メモリボードのローページの大きさと、16K バイトの EMS 標準ページの大きさとの違いは、ファンクション 43H で得られるハンドルを、そのアプリケーションが使用しているときに明白になります。

メモリマネージャは、ハンドルにアロケートされたページと、ローハンドルにアロケートされたページを区別しなければなりません。ドライバへの呼び出しの意味は、メモリマネージャに、ハンドルが渡されるかローハンドルが渡されるかによって変わります。たとえば、ハンドルがファンクション 51H に渡されれば、メモリマネージャはハンドルに割り当てられた 16K バイトのページ数を増加／減少させます。もし、ファンクション 51H がローハンドルを渡されれば、メモリマネージャはローハンドルに割り当てられたローページ（16K バイトではない）を増加／減少させます。ただし、EMS 標準ボードにおいては、ページとローページの違いはありません。

システムローハンドル（ローハンドル＝ 0000H）

640K バイトよりも、下のアドレス空間にあるメモリをマップし直すことが可能な拡張メモリボード

では、640K以下の部分に位置するメモリのページを管理することが、ひとつの問題になります。メモリマネージャはこの問題を解決するために、MS-DOSがマネージャをロードするときに値0000Hを持つローハンドルを作ります。このローハンドルは、システムハンドルと呼ばれます。

さらに極端な場合、メモリマネージャは640Kバイト以下にマップされる全ページをシステムハンドルに割り当てます。これらのページは、論理的順序でマップされなければなりません。たとえば、システムボードが256KバイトのRAMをサポートし、それよりも384Kバイト上位がマップ可能であれば、そのシステムハンドルはその論理ページ0を、256Kにある第1物理ページに、論理ページ1を次の物理ページにというようにマップしたほうがよいでしょう。

システムハンドルは、ローページを扱わなければなりません。アプリケーションが使用できるように、これらのページを何ページか開放するために、OSが再アロケートファンクションを使用して、システムハンドルに割り当てられたページ数を減少させることができます。デアロケートファンクションを実行すると、システムハンドルの大きさは0になりますが、ローハンドル自体のアロケートを解除するわけではありません。デアロケートファンクションは、他のローハンドルとシステムハンドルを区別して扱います。もしOSが以前に〝exited〟されているならば（たとえばMS-WINDOWSが〝exited〟しているように)、元のシステムハンドルの640Kバイトを満たすように増加させ、DOSに返す前に元の論理ページを物理メモリにマップします。

また、このハンドルには機能上、次のような特別な場合があります。

1. ファンクション43H（ページのアロケート）を扱います。このファンクションはハンドルの値として0を返すことはありません。アプリケーションはいつもファンクション43Hを呼び出して、ページをアロケートし、ページを取り扱うハンドルを取得しなければなりません。決してハンドル値0を返さないので、アプリケーションはこの特別なハンドルへアクセスすることはできません。

2. ファンクション45H（ページのデアロケート）を扱います。もし、OSがこのファンクションを使用して、システムハンドルにアロケートされているページを解除すると、そのページはマネージャに返され、再利用できます。ただし、ハンドルを再び割り当てることはできません。マネージャはこのハンドルに対するデアロケートファンクションの要求を、再アロケートファンクションの要求と同じものとして扱います。このハンドルに割り当てられるページ数は0です。

5.7　拡張メモリマネージャ有無のテスト

アプリケーションが拡張メモリマネージャを使用する前に、MS-DOSは拡張メモリマネージャをロードしたかどうかを調べなければなりません。ここでは、プログラム中で、メモリマネージャの有無を調べるための2つの方法を説明します。いずれもMS-DOSのファンクションリクエストを利用するもので、1つはファンクション3DH（ハンドルを使うファイルのオープン）で、もう1つはファンクション35H（割り込みベクタの取得）です（ファンクションリクエストについては「MS-DOSプログラマーズリファレンスマニュアルVol.1」を参照してください)。ここでは、どのような状況でどちらの方法が適切であるかを説明します。

ほとんどのアプリケーションでは、ファンクション3DH（ハンドルを使うファイルのオープン）もファンクション35H（割り込みベクタの取得）も使用することができます。しかし、次の2つの場合は

ファンクション 35H（割り込みベクタの取得）を使用しなければなりません。

- 扱うプログラムが、デバイスドライバになる場合
- ファイルシステム操作中に MS–DOS に割り込みをかける場合

デバイスドライバは、MS–DOS 内部から実行されるので、MS–DOS ファイルシステムをアクセスすることができません。ファイルシステムの操作中に MS–DOS に割り込みをかけるプログラムでも、類似した制限があります。割り込み処理の手続きが行われている間は、プログラムは MS–DOS ファイルシステムにアクセスすることができません。なぜならば、他のプログラムが、そのファイルシステムを必要としないからです。

次にそれぞれの使用方法について説明します。

■ ファンクション 3DH（ハンドルを使うファイルのオープン）

大部分のアプリケーションは、MS–DOS のファンクション 3DH（ハンドルを使うファイルのオープン）を使用して、メモリマネージャの有無を調べることができます。ここではその方法をプログラム例とともに説明します。

注意　扱うプログラムが、デバイスドライバになるか、ファイルシステムの操作中に MS–DOS に割り込みをかける場合は、この方法を使用しないでください。このようなプログラムでは、ファンクション 35H（割り込みベクタの取得）を使用してください。

●ファンクション 3DH（ハンドルを使うファイルのオープン）の使用方法

ここでは、ファンクション 3DH（ハンドルを使うファイルのオープン）を使用して、メモリマネージャの有無を調べる方法を説明します。この方法は、次の手順で行います。

1. 〝read only〞のアクセスモード（レジスタ AL = 0）で、ファンクション 3DH を実行します。このファンクションは、目的とするファイルやデバイスのパス名を含む ASCII 文字列を指示するためのプログラムが必要です（レジスタセット DS：DX がポインタを含む）。この場合には、そのファイルは実際のメモリマネージャの名前です。
   次のような形式の ASCII 文字列を使用します。

   ASCII device name DB 〝EMMXXXX0〞,0

   大文字 EMMXXXX0 に対する ASCII コードは 00H（NULL）で終わります。

2. MS–DOS がエラーコードを返さない場合は、2、3 を省略して 5 へ進みます。エラーが〝Too many open files〞であった場合は、3 へ進みます。エラーが〝File/Path not found〞の場合は、3 を省略して 4 へ進みます。

3. MS–DOSがハンドルが十分でないことを示す〝Too many open handle〞エラーを返した場合には、他のファイルをオープンする前に、プログラムでファイルのクローズ（MS–DOSファンクション10H）を実行します。これによって、少なくとも1つのファイルハンドルをこのエラーにならずに、このファンクションの実行で利用できるようになります。
   プログラムがファンクション3DHを実行した後では、6で説明するテストを実行し、ハンドルをクローズします（MS–DOSファンクション3EH）。〝manager〞ファンクションはMS–DOSを通じて利用することができないので、このステータステストの実行後も、そのマネージャをオープンしたままにしてはいけません。この場合は6へ進みます。

4. MS–DOSが〝File/Path not found〞エラーを返した場合には、そのメモリマネージャはインストールされていません。アプリケーションがメモリマネージャを必要としているのであれば、メモリマネージャと適切なCONFIG.SYSファイルを収めているディスクでシステムをブートしなおさなければなりません。

5. MS–DOSがエラーステータスコードを返さない場合は、〝EMMXXXX0〞という名前のデバイスがシステムに常駐しているか、同じ名前のファイルが現在のディスクドライブ装置内に存在していると考えられます。この場合は6へ進みます。

6. IOCTLデータを得るファンクション（MS–DOSファンクション4400H）を実行し、EMMXXXX0がデバイスかファイルかを調べます。このとき、〝EMM〞デバイスにアクセスするために、1で取得したファイルハンドル（レジスタBX）を使用します。このファンクションは、レジスタDXにデバイス情報を返します。次に7へ進みます。

7. MS–DOSがなんらかのエラーステータスコードを返す場合には、メモリマネージャデバイスドライバがインストールされていないことが考えられます。もし、アプリケーションがメモリマネージャを必要とするのであれば、メモリマネージャと適切なCONFIG.SYSファイルを収めているディスクで、システムをブートしなおさなければなりません。

8. もしMS–DOSがエラーステータスを返さなければ、ファンクション4400Hが返したデバイス情報（レジスタDX）の第7ビット（0から数えて）を調べます。次に9へ進みます。

9. もしデバイス情報の第7ビットが0であったならば、〝EMMXXXX0〞はファイルであり、メモリマネージャデバイスドライバは存在しません。もし、アプリケーションがメモリマネージャを必要とするのであれば、メモリマネージャと適切なCONFIG.SYSファイルを収めているディスクで、

システムをブートしなおさなければなりません。
もし第7ビットが1であったならば、EMMXXX0はデバイスです。この場合は10へ進みます。

10. 出力ステータスを得るファンクション（MS–DOSファンクション4407H）を実行します。
ユーザーは、〝EMM〞デバイスにアクセスするために、1で取得したファイルハンドル（レジスタBX）を使用しなければなりません。次に11へ進みます。

11. もし、拡張メモリマネージャが〝ready〞ならば、メモリマネージャはレジスタALにステータス値〝0FFH〞を渡します。もし、デバイスドライバが〝not ready〞ならば、ステータス値は〝00H〞です。
もしメモリマネージャデバイスドライバが〝not ready〞で、アプリケーションがそれを必要としているならば、メモリマネージャと適切なCONFIG.SYSファイルを収めたディスクでシステムをブートしなおさなければならないでしょう。
もし、メモリマネージャデバイスドライバが〝ready〞であれは、12へ進みます。

12. 拡張メモリデバイスドライバをクローズするために、ファンクション3EHを実行します。このとき〝EMM〞デバイス（レジスタBX）をクローズするために、1で取得したファイルハンドルを使います。

●ファンクション3DH（ハンドルを使うファイルのオープン）のプログラム例

次の手続きは、前述のファンクション3DH（ハンドルを使うファイルのオープン）のプログラム例です。

システム中のEMMの存在を調べます。これを実行すると、EMMが存在すればキャリーフラグがセットされ、存在しなければキャリーフラグはクリアされます。

```
first_test_for_EMM     PROC NEAR
PUSH   DS
PUSH   CS
POP    DS
MOV    AX,3D00H         ; 〝read only〞モードで〝device open〞を実行する
LEA    DX,ASCII_device_name
INT    21H
JC     first_test_for_EMM_error_exit
                        ; 〝device open〞中のエラーを調べる
MOV    BX,AX            ;DOSによって返された〝file handle〞を得る
MOV    AX,4400H         ; 〝IOCTL〞を実行する
```

```
INT    21H              ; "get device info."
JC     first_test_for_EMM_error_exit
                        ; "get device info." 中のエラーを調べる
TEST   DX,0080H         ;ASCII device name がデバイスかファイルかを調べる
JZ     first_test_for_EMM_error_exit
MOV    AX,4407H         ; "IOCTL" を実行する
INT    21H
JC     first_test_for_EMM_error_exit
                        ; "IOCTL" 中のエラーを調べる
PUSH   AX               ; "IOCTL" ステータスを保存する
MOV    AH,3EH           ; "CLOSE FILE HANDLE" を実行する
INT    21H
POP    AX               ; "IOCTL" ステータスを復元する
CMP    AL,0FFH          ; ドライバによって返された "DEVICE READY"
                        ; ステータスを調べる
JNZ  first_test_for_EMM_error_exit
first_test_for_EMM_exit：
POP   DS                ;EMM がシステムに存在する
STC
RET
first_test_for_EMM_error_exit：
POP   DS                ;EMM はシステムに存在しない
CLC
RET
ASCII_deice_name        DB"EMMXXXX0",0
first_test_for_EMM      ENDP
```

■ファンクション 35H（割り込みベクタの取得）

大部分のアプリケーションは、MS–DOS のファンクション 35H（割り込みベクタの取得）を使用して、メモリマネージャの有無を調べることができます。ここではその方法をプログラム例とともに説明します。

注意　扱うプログラムがデバイスドライバで、ファイルシステムの操作中にMS–DOS に割り込みをかける場合は必ずこの方法を使用してください。

●ファンクション 35H（割り込みベクタの取得）の使用方法

ここでは、メモリマネージャの有無を調べる MS–DOS のファンクション 35H（割り込みベクタの取得）を説明します。この方法は、次の手順で行います。

1. ファンクション 35H を実行して、割り込みベクタのエントリ番号 67H の内容を取得します（0000：019C～0000：019F のアドレス）。
   メモリマネージャは、どのマネージャファンクションを実行する場合でも、

この割り込みベクタを使用します。この割り込みサービスルーチンのアドレスのオフセット部は、アドレス 0000：019C にあるワード中に、セグメント部はアドレス 0000：019E にあるワード中に保存されます。

2. 割り込みベクタアドレス 67H のセグメント部と、固定オフセット 000AH で指定されるアドレスで始まる ASCII 文字列の内容と、デバイス名フィールドとを比較します。もし MS-DOS が、ブート時にメモリマネージャをロードしていれば、このフィールドにはデバイス名が入ります。
メモリマネージャは、キャラクタデバイスドライバとしてインプリメントされているので、プログラムは 0000H から始まります。デバイスドライバは、その位置に位置づけられたデバイスヘッダを持っています。デバイスヘッダには、8 バイトのデバイス名フィールドがあります。キャラクタデバイスドライバにおいては、このフィールドはいつもデバイスヘッダ内のオフセット 000AH に位置づけられます。このフィールドは、MS-DOS がデバイスを参照するときに、そのデバイス名を含んでいます。
もし、この方法で〝文字列比較〟の結果が正しければ、メモリマネージャデバイスは存在します。

●ファンクション 35H（割り込みベクタの取得）のプログラム例

次の手続きは、前述のファンクション 35H（割り込みベクタの取得）のプログラム例です。

システム中の EMM の存在を調べます。これを実行すると、EMM が存在すればキャリーフラグがセットされ、存在しなければキャリーフラグはクリアされます。

```
second_test_for_EMM     PROC NEAR
PUSH   DS
PUSH   CS
POP    DS
MOV    AX,3567H       ;〝get interrupt vector〟を実行する
INT    21H
MOV    DI,000AH       ;DOS で返された ES の SEGMENT を使用する。
                      ;DI での〝device name field〟オフセットを配置する
LEA    SI,ASCII_device_name
                      ;SI で装置名文字列のオフセットを配置する
                      ;SEGMENT はすでに DS 内にある
MO     CX,8           ; 名前文字列を比較する
CLD
REPE   CMPSB
JNE    second_test_for_EMM_error_exit
second_test_for_EMM_exit：
POP    DS             ;EMM がシステムに存在する
```

```
STC
RET
second_exit_for_EMM_exit：
POP    DS              ;EMMはシステムに存在しない
CLC
RET
ASCII_device_name       DB"EMMXXXX0"
second_test_for_EMM     ENDP
```

5.8　ファンクション5B00H〜5B08HにおけるOSの利用

拡張メモリボードには、論理ページから物理ページへのマップを〝記憶する〟1組のレジスタがあります。いくつかのボードには、数組の特別な（代替用）マップレジスタがあります。拡張メモリボードは、代替用マップレジスタの組を限られた数しか提供することはできないので、この仕様ではファンクション5B00H〜5B08H（マップレジスタの変更）を使用して、それらをシミュレートする方法を提供しています。

●プログラム例

ここでの例は、ハードウェアか代替用マップレジスタセットをサポートしていことを想定しています。第1ウィンドウを持ってきて、それから〝Reversi〟が始まります。このとき制御はMS-DOS監視プログラムに返り、切り換わります。この手続きのために拡張メモリマネージャへの呼び出しがあります。

例1

```
Allocate alt reg set                ;MS-DOSをスタートする
(for the MS-DOS executive)          ;監視プログラム
Set all reg set
(for MS-DOS Executive)
Allocate alt reg set                ;Reversiをスタートする
(for Reversi)
Set alt reg set
(for Reversi)
Map pages
(for Reversi)
Set alt reg set                     ;MS-DOSへ返って切り替えられる
(for MS-DOS Executive)              ;監視プログラム
```

例2

```
Allocate alt set                    ;MS-DOSをスタートする
(for the MS-DOS Executive)          ;監視するプログラム
Set alt reg set
```

```
(for MS-DOS Executive)
Get alt reg set
(for MS-DOS Executive)
Allocate alt reg set              ;Reversi をスタートする
(for Reversi)
Set alt reg set
(for Reversi)
Map pages
(for Reversi)
Get alt reg set
(for Reversi)
Set alt reg set                   ;MS-DOS へ返って切り換える
(for MS-DOS Executive)            ; 監視プログラム
```

ここで注意しなければならない点は、Set（設定）は必ず Get（取得）に先行しなければならないということです。Set のあとの Get という形式は、割り込みハンドルが使用する Get のあとで Set という形式（古いマッピングコンテキストを得て、新しいものを設定する）の逆になります。また、代替用マップレジスタの設定は、割り当てられるとき、現在のマップを持たなければならず、そうでないときはその Set は混乱をきたすことになります。

これがソフトウェアでシミュレートされれば、同じ Set と Get のモデルが当てはまります。大きな違いは、コンテキストが保存される場所です。

アロケート呼び出しが動的で、割り当てられている組数に制限がないので、OS は要求される空間を提供しなければなりません。しかしデバイスドライバは、動的に空間をアロケートすることができません。もしアロケートレジスタセットの呼び出しが、代替用マップレジスタセットをサポートしていないというステータスを返した場合、OS はそのコンテキストに対して空間をアロケートしなければなりません。また、ファンクション 4E00H を用いてコンテキストを初期化しなければなりません。その時点で OS は、Set をマッピングコンテキスト空間に対するポインタを渡すことで、実行することができます。Get の呼び出しに関しては、EMM デバイスドライバにその同じコンテキストの空間に対するポインタを返します。

例 3

```
Allocate alt reg set              ;MS-DOS をスタートする
(for the MS-DOS executive)        ; 監視プログラム
Get page Map
(for MS-DOS executive)
Set alt reg set
(for MS-DOS Executive)
Allocate alt reg set              ;Reversi をスタートする
(for reversi)
Set alt reg set
(for Reversi)
Map pages
```

```
(for Reversi)
Get Page Map
(for Reversi)
Set alt reg set                ;MS-DOS へ返って切り替える
(for MS-DOS Executive)         ; 監視プログラム
```

5.9 用語

ここでは、この解説で数多く使用された用語をいくつか説明します。

EMM（イーエムエム）

拡張メモリマネージャを参照。

拡張メモリ（Extended Memory）

保護仮想アドレスモードで操作されているとき、80286、386 / 386SX、486 / 486SXプロセッサ上で利用可能な100000HからFFFFFFHまでの15Mバイトの範囲アドレスを持つメモリ。PC–9800シリーズの〝拡張メモリ〟はこのExtended Memoryのことを指す。

拡張メモリマネージャ（Expanded Memory Manager、EMM）

MS–DOSのアプリケーションと拡張メモリのインターフェイスを制御するデバイスドライバ。

ローページ（Raw Page）

拡張メモリボードが与えることができる、マップ可能な最小単位のメモリ。

常駐アプリケーション（Resident Application Program）

常駐アプリケーションはMS–DOSによってロードされ、実行され、MS–DOSに制御を戻した後でも、システムに常駐するプログラムのこと。このタイプのプログラムはメモリを占有し、通常、OSやアプリケーションやハードウェアによって呼び出される。常駐アプリケーションの例としては、RAMディスクドライバ、プリントスプーラ、〝ポップアップ〟デスクトッププログラムなどがあげられる。

通常メモリ（Conventional Memory）

アドレス00000Hから9FFFFHにある、0〜640Kバイトのメモリ。

PC–9800シリーズのハイレゾリューションモードでは、00000H〜BFFFFHの768Kバイトのメモリになる。

アロケート（Allocate）

拡張メモリのページを、指定した大きさだけ確保すること。

デアロケート（Deallocate）

以前にアロケートした拡張メモリを、メモリマネージャに返すこと。

ハンドル（Handle）
　EMMが、アプリケーションで用いられる1ブロックのメモリを識別するために割り当てて使用する値。割り当てられたすべての論理ページは、個々のハンドルと関連がある。

非常駐アプリケーション（Transient Application Program）
　非常駐アプリケーションは、MS–DOSによってロードされ、実行され、制御をMS–DOSに返した後に、そのシステムには常駐しないプログラムのこと。非常駐アプリケーションがMS–DOSに制御を移した後は、そのプログラムが使用していたメモリを他のプログラムが利用できる。

物理ページ（Physical Page）
　隣接した16Kバイトの物理ページの集まりで、アプリケーションはこれを介して拡張メモリをアクセスする。

ページフレーム基底アドレス（Page Frame Base Address）
　ページフレーム基底アドレスは、ページフレームの最初のバイトの位置（セグメント形式）である。

マッピング（Mapping）
　物理ページ上に現れる、メモリの論理ページを作る処理。

マッピングコンテキスト（Mapping Context）
　ある特定の時点でのマップレジスタの内容。このコンテキストは、マップの状態を表している。

マップ解除（Unmap）
　論理ページに対して、読み込み／書き出しのアクセスを不可能にすること。

マップ可能なセグメント（Mppable Segment）
　マップされた論理ページを持つことが可能な、16Kバイトのメモリ領域。

マップレジスタ（Map Registers）
　EMMハードウェアの現在のマッピングコンテキストを含んでいる1組のレジスタ。

論理ページ（Logical Page）
　EMMは、拡張メモリを論理ページと呼ばれる（典型的には16Kバイト）単位にアロケートする。

5.10　ファンクションとステータスコードのクロスリファレンス

ここでは、2つのクロスリファレンスを掲載しています。1つは、ファンクションコードと、それが返すステータスコードとの対応を示す一覧表です。もう1つは、ステータスコードと、それらを返すファンクションとの関係を示す一覧表です。

■ファンクションとステータスコードのクロスリファレンス

ファンクション	ステータス（16進）	説　明
40H	00H,80H,81H,84H	ステータスの取得
41H	00H,80H,81H,84H	ページフレームのアドレスの取得
42H	00H,80H,81H,84H	未アロケートページ数の取得
43H	00H,80H,81H,84H,85H,87H,88H,89H	ページのアロケート
44H	00H,80H,81H,83H,84H,8AH,8BH	ハンドルページのマップ／アンマップ
45H	00H,80H,81H,83H,84H,86H	ページのデアロケート（開放）
46H	00H,80H,81H,84H	バージョンの取得
47H	00H,80H,81H,83H,84H,8CH,8DH	ページマップのセーブ
48H	00H,80H,81H,83H,84H,8EH	ページマップのリストア
49H		システム予約
4AH		システム予約
4BH	00H,80H,81H,84H	ハンドル数の取得
4CH	00H,80H,81H,83H,84H	ハンドルページの取得
4DH	00H,80H,81H,84H	全ハンドルページの取得
4E00H	00H,80H,81H,84H,8FH	ページマップの取得
4E01H	00H,80H,81H,84H,8FH,A3H	ページマップの設定
4E02H	00H,80H,81H,84H,8FH,A3H	ページマップの取得と設定
4E03H	00H,80H,81H,84H,8FH	ページマップセーブ配列のサイズ取得
4F00H	00H,80H,81H,84H,8BH,8FH,A3H	ページマップの一部をセーブ
4F01H	00H,80H,81H,84H,8FH,A3H	ページマップの一部をリストア
4F02H	00H,80H,81H,84H,8BH,8FH	ページマップの一部をセーブする配列のサイズ取得

ファンクション	ステータス（16進）	説　明
5000H	00H,80H,81H,83H, 84H,8AH,8BH,8FH	複数ハンドルページのマップ／アンマップ (論理ページ／物理ページ方式)
5001H	00H,80H,81H,83H, 84H,8AH,8BH,8FH	複数ハンドルページのマップ／アンマップ (論理ページ／セグメントアドレス方式)
51H	00H,80H,81H,83H, 84H,87H,88H	ページの再アロケート
5200H	00H,80H,81H,83H, 84H,8FH,91H	ハンドルアトリビュートの取得
5201H	00H,80H,81H,83H, 84H,8FH,90H,91H	ハンドルアトリビュートの設定
5202H	00H,80H,81H,84H, 8FH	ハンドルアトリビュートのケイパビリティの取得
5300H	00H,80H,81H,83H, 84H,8FH	ハンドル名の取得
5301H	00H,80H,81H,83H, 84H,8FH,A1H	ハンドル名の設定
5400H	00H,80H,81H,84H, 8FH	ハンドルのディレクトリ取得
5401H	00H,80H,81H,84H, 8FH,A0H,A1H	指定ハンドルのサーチ
5402H	00H,80H,81H,84H, 8FH	ハンドルの総数の取得
55H	00H,80H,81H,83H, 84H,8AH,8BH,8FH	ページマップの変更とジャンプ
56H	00H,80H,81H,83H, 84H,8AH,8BH,8FH	ページマップの変更とコール
5602H	00H,80H,81H,84H, 8FH	ページマップスタックサイズの取得
5700H	00H,80H,81H,83H, 84H,8AH,8FH,92H, 93H,94H,95H,96H, 98H,A2H	メモリ領域の移動
5701H	00H,80H,81H,83H, 84H,8AH,8FH,93H, 94H,95H,96H,97H, 98H,A2H	メモリ領域の交換
5800H	00H,80H,81H,83H, 84H,8FH	マップ可能な物理アドレス配列の取得
5801H	00H,80H,81H,84H, 8FH	マップ可能な物理アドレス配列エントリの取得
5900H	00H,80H,81H,83H, 84H,8FH,A4H	ハードウェア構成配列の取得
5901H	00H,80H,81H,83H, 84H,8FH	未アロケートのローページ数の取得

ファンクション	ステータス（16進）	説　明
5A00H	00H,80H,81H,84H, 85H,87H,88H,8FH	標準サイズのページのアロケートと固有のEMMハンドルの割り当て
5A01H	00H,80H,81H,84H, 85H,87H,88H,8FH	ローページのアロケートと固有のEMMハンドルの割り当て
5B00H	00H,80H,81H,83H, 84H,8FH,A4H	代替マップレジスタセットの取得
5B01H	00H,80H,81H,83H, 84H,8FH,9AH,9CH, 9DH,A3H,A4H	代替マップレジスタセットの設定
5B02H	00H,80H,81H,83H, 84H,8FH,A4H	代替マップセーブ配列のサイズ取得
5B03H	00H,80H,81H,83H, 84H,8FH,9BH,A4H	代替マップレジスタセットのアロケート
5B04H	00H,80H,81H,83H, 84H,8FH,9CH,9DH A4H	代替マップレジスタセットの開放
5B05H	00H,80H,81H,83H, 84H,8FH,9BH,A4H	DMAレジスタセットのアロケート
5B06H	00H,80H,81H,83H, 84H,8FH,9AH,9CH, 9DH,9EH,9FH,A4H	代替マップレジスタ上のDMAの使用許可
5B07H	00H,80H,81H,83H, 84H,8FH,9AH,9CH, 9DH,9EH,9FH,A4H	代替マップレジスタ上のDMAの使用不許可
5B08H	00H,80H,81H,83H, 84H,8FH,9CH,9DH, A4H	DMAレジスタセットの開放
5CH	00H,80H,81H,83H 84H	ウォームブートのための拡張メモリの準備
5D00H	00H,80H,81H,84H, 8FH,A4H	OS/Eファンクションセットの使用許可
5D01H	00H,80H,81H,84H, 8FH,A4H	OS/Eファンクションセットの使用不許可
5D02H	00H,80H,81H,84H, 8FH,A4H	アクセスキーのリターン
7000H	00H,80H	ページフレーム用バンクのステータスの設定
7001H	00H,80H	ページフレーム用バンクの状態の設定

■ ステータスとファンクションコードのクロスリファレンス

ステータス	ファンクション	説　明
00H	全部	ファンクションは正常に終了した。
80H	全部	メモリマネージャが、拡張メモリソフトウェアの障害を見つけた。もし、メモリマネージャが正常に操作されていれば起こらない状況が発見された。
81H	全部	メモリマネージャが、拡張メモリハードウェアの障害を見つけた。もしメモリハードウェアが正常に動作していれば起こらない状況が発見された。その原因を調べるために拡張メモリシステムまで診断するべきである。
82H	なし	このエラーコードは、メモリマネージャのバージョン3.2以上のバージョンでは返されない。メモリマネージャの初期バージョンでは、このコードは〝busy〞ステータスを意味していた。このステータスは、現在の要求が発生したときにすでに拡張メモリ要求を処理中で、他の要求を処理できないことを示している。メモリマネージャの3.2以上のバージョンでは、メモリマネージャは決して〝busy〞にはならず、いつでも要求を受け入れることができる。
83H	44H,45H,47H,48H,4CH,5000H,5001H,51H,5200H,5201H,5300H,5301H,5400H,55H,56H,5700H,5701H,5800H,5900H,5901H,5B00H,5B01H,5B02H,5B03H,5B04H,5B05H,5B06H,5B07H,5B08H,5CH	メモリマネージャが、指定したハンドルを見つけることができない。プログラムが、指定したハンドルを壊したかもしれない。メモリマネージャは、指定したハンドルに関する情報を持っていない。
84H	全部	マネージャに渡されたファンクションコードは、現在定義されていない。現在、定義されているファンクションは、40H〜5DHと7000Hである。
85H	43H,5A00H,5A01H	現在利用できるハンドルが存在しない。現在、すべての割り当て可能なハンドルが使用されている。プログラムは、他のプログラムがハンドルを開放するのを待って、ハンドルの割り当てを再要求することができる。ハンドルは255個までサポートできる。

ステータス	ファンクション	説　明
86H	45H	マッピングコンテキスト復元エラーが発見された。このエラーは、プログラムがハンドルを返すことを試み、指示されたハンドルに対するコンテキストスタックに〝マッピングコンテキスト〟がまだあるときに生じる。プログラムは、ハンドルを返す前にマッピングコンテキストを復元することによって、このエラーから回復することができる。
87H	43H,51H,5A00H,5A01H	システム中で利用できるページ数が、新たなアロケート要求には不十分である。プログラムは、より少ないページ数を要求することで、この状態から回復できる。
88H	43H,51H,5A00H,5A01H	現在利用可能なアロケートされていないページ数では、割り当て要求を受け入れるには不十分である。プログラムは、利用可能なページ数が十分になったとき、もう一度要求するか、より少ないページを要求することによってこの状態から回復できる。
89H	なし	このエラーコードはメモリマネージャのバージョン4.0か、それ以後のバージョンでは返されない。メモリマネージャの初期バージョンでは、このコードは0ページがハンドルに割り当てることができないことを示した。現在この状態は存在しない。
8AH	44H,5000H,5001H,55H,56H,5700H,5701H	メモリにマップする論理ページが、ハンドルに割り当てられている論理ページの範囲外にある。プログラムは、ハンドルの限界内にある論理ページをマップすることによって、この状態から回復できる。
8BH	44H,4F00H,4F02H,5000H,5001H,55H,56H	1ページ以上の物理ページが、許容可能な分離ページの範囲外にある。物理ページの数は相対的に0から番号づけされている。プログラムは、システムがサポートしている物理ページにマップすることによってこの状態から回復できる。
8CH	47H	マップレジスタコンテキスト保存領域がいっぱいである。プログラムは再びマップレジスタを保存することを試みることによって、この状態から回復できる。
8DH	47H	マップレジスタコンテキストスタックには、すでにハンドルに関連したコンテキストが存在する。プログラムは、ハンドルに対するコンテキストがスタックにすでに存在しているとき、マップレジスタコンテキストを保存することを試みた。プログラムは再びコンテキストを保存しなければ、この状態から回復できる（これは、ハンドルのスタック上のマップレジスタコンテキストが正しいことを仮定している）。

ステータス	ファンクション	説　明
8EH	48H	マップレジスタコンテキストスタックには、指定されたハンドルに関係があるコンテキストは存在しない。プログラムは、ハンドルに対するコンテキストがスタックに存在しないとき、マップレジスタコンテキストを復元しようとした。プログラムは再びコンテキストを復元しようとしなければ、この状態から回復できる（これは、現在のマップレジスタコンテキストが正しいことを仮定している）。
8FH	部分ファンクションコードを必要としている、すべてのファンクション	ファンクションに渡される部分ファンクションパラメータが定義されていない。
90H	5201H	属性の型が未定義である。
91H	5200H,5201H	システム構成が、不揮発性をサポートしていない。
92H	5700H	拡張メモリの、ソース領域とコピー先の領域が同じハンドルを持ち、重なりあっている。この場合にもコピーは有効である。コピーが完成すると、少なくともソース領域の一部がコピーによって上書きされてしまう。異なるハンドルをもつ拡張メモリのソース領域とコピー先の領域は、決して物理的に重なり合わないように注意すること。なぜなら異なるハンドルは、拡張メモリの異なる領域を指定するため。
93H	5700H,5701H	拡張メモリの指定されたソース領域か、コピー先の領域の大きさが、相手の領域より大きい。指定された大きさの領域をコピー／交換するには、このハンドルにアロケートされたページでは不十分である。プログラムは、コピー先かコピー元のハンドルにページを追加してアロケートするか、または指定した大きさを減らすことによって、この状態から回復する。しかし、もしアプリケーションに、それが必要とするメモリと同じ大きさの拡張メモリが割り当てられていれば、それはプログラムエラーであり、回復不可能である。
94H	5700H,5701H	標準メモリ領域と拡張メモリ領域が、重なりあっている。これは無効である。標準メモリ領域は、拡張メモリ領域と重なり合うことができない。
95H	5700H,5701H	論理ページ内にあるオフセットが、論理ページの大きさよりも大きい。拡張メモリ領域内にあるコピー元やコピー先のオフセットは、0から（論理ページ-1）、あるいは0から16383（3FFFH）の間になければならない。
96H	5700H,5701H	領域の大きさが1Mバイトを超えている。

ステータス	ファンクション	説　明
97H	5701H	拡張メモリのソース領域と交換先の領域が同じハンドルを持ち、重なり合っている。これは無効である。拡張メモリのソース領域と交換先の領域が交換されているとき、同じハンドルを持って重なり合うことができない。異なったハンドルをもつ拡張メモリソース領域と交換先の領域は、決して物理的に重なり合わないように注意すること。なぜなら異なったハンドルは拡張メモリの異なった領域を指定するため。
98H	5700H,5701H	元のメモリと目的のメモリの型が未定義である／サポートされていない。
9AH	5B01H,5B06H, 5B07H	代替用マップレジスタセットがサポートされているが、指定された代替用マップレジスタはサポートされていない。
9BH	5B03H,5B05H	代替用マップ／DMAレジスタセットはサポートされているが、すべての代替用マップ／DMAレジスタセットが現在アロケートされている。
9CH	5B01H,5B04H, 5B06H,5B07H, 5B08H	代替用マップ／DMAレジスタセットはサポートされていない。また指定された代替用マップ／DMAレジスタセットは0でない。
9DH	5B01H,5B04H, 5B06H,5B07H, 5B08H	代替用マップ／DMAレジスタセットはサポートされているが、指定された交替用マップレジスタセットが定義、またはアロケートされていないか、すでにアロケートされているマップレジスタセットである。
9EH	5B06H,5B07H	専用のDMAチャネルがサポートされていない。
9FH	5B06H,5B07H	専用のDMAチャネルはサポートされているが、指定したDMAチャネルはサポートされていない。
A0H	5401H	指定したハンドル名のハンドル値が見つからない。
A1H	5301H,5401H	この名前をもつハンドルはすでに存在している。指定されたハンドルは名前が割り当てられていない。
A2H	5700H,5701H	コピー／交換中に1Mバイトのアドレス空間を超えようとした。コピー／交換する領域の大きさまたはソースの先頭アドレスが1Mバイトを超えている。データはコピー／交換されていない。
A3H	4E01H,4E02H, 4F00H,4F01H, 5B01H	ファンクションに渡されたデータ構造の内容が不正に取り扱われているか、意味のないものである。
A4H	5900H,5B00H, 5B01H,5B02H, 5B03H,5B04H, 5B05H,5B06H, 5B07H,5B08H, 5D00H,5D01H, 5D02H	OSが、このファンクションのアクセスを拒否した。このファンクションはこの時点で使用できない。

第6章

フォントドライバ

6.1 イントロダクション

フォントドライバは、マルチフォント ROM ボードや本体 ROM、本体マルチフォントなどを利用して、2バイト JIS コードの文字フォントを編集したり、利用者定義フォントを拡大、縮小し、ユーザー領域に出力するデバイスドライバです。

得られたフォントは、グラフィックスドライバを利用することにより、グラフィック VRAM に出力することができます。

MS-DOS では、フォントを制御するソフトウェアを〝FONT.SYS〟というデバイスドライバで提供しています。フォントの機能を利用するには、〝FONT.SYS〟を CONFIG.SYS ファイルに組み込みます。デバイスドライバの詳細な組み込み方法についてはユーザーズリファレンスマニュアルを参照してください。

注意　PC-9801、PC-9801E、PC-9801F、PC-9801M ではマルチフォント ROM ボードが使用できないので、本体 ROM のみのサポートとなります。また、PC-98LT、PC-98HA では動作しません。

6.2　文字フォントの利用方法

フォントドライバで取得した文字フォントをグラフィック VRAM に描画するには、グラフィックスドライバを利用します。

フォントドライバの文字フォントのデータ形式は、グラフィックスドライバの〝グラフィックイメージの設定〟で利用するモノクロモードのグラフィックイメージデータと同一になっています。したがって、取得した文字フォントのアドレスを格納域アドレス（グラフィックスドライバのパラメータ）に指定して、グラフィックスドライバのファンクション No.24（グラフィックイメージの設定）を実行することにより、文字フォントをグラフィック VRAM に描画します。

6.3　フォントファンクションの呼び出し方法

フォントドライバの各ファンクションは次の手順で呼び出します。

1. フォントドライバのエントリテーブルの先頭アドレスを取得します（エントリテーブルの詳細はフォントファンクション一覧を参照してください）。
   AX レジスタ、DS：BX レジスタを次のようにセットし、割り込みタイプ CCH（INT CCH）を行うとエントリテーブルの先頭アドレスが、指定したアドレスに設定されます。

   AX　　＝0
   DS：BX＝先頭アドレスを格納する領域（ダブルワード）のアドレス

2. ファンクションの呼び出します。
   パラメータブロック（32 バイト）に必要なパラメータを設定し、この先頭アドレスをスタックに格納後、エントリテーブル内の対応するアドレスに far call します。詳細は各ファンクションの解説を参照してください。

データ領域のアドレス（DWORD）→

パラメータブロック 32 バイト

なおプログラムの使用例については 6.6「プログラム例」を参照してください。

6.4 フォントファンクション一覧

フォントデバイスドライバには次のような機能が用意されています。

また、各ファンクションのエントリテーブルの詳細もあわせて示しています。各テーブルにはそのファンクションのエントリアドレスのが格納されています。

ファンクションNo	アドレス	機　能
0	0000	バージョンの取得
1	0004	フォント情報の設定
2	0008	フォント情報の取得
3	000C	フォントの取得

これ以降では、各ファンクションごとにその使用方法を解説します。

INT CCH

ファンクション
No.0 バージョンの取得

コール　スタック＝パラメータブロックの先頭アドレス

リターン　AX ＝フォントドライバのバージョン
AL　バージョン番号の整数部
AH　バージョン番号の小数部

解　説

フォントドライバのバージョンを取得します。

たとえば、バージョン2.0ではAXに0002Hが入ります。

このファンクションで、バージョンに1.0（AX＝0001H）が返された場合、ファンクションNo.1（フォント情報の設定）で動作指定フラグに0以外を設定することはできません。

INT CCH

ファンクション No.1 フォント情報の設定

コール

スタックニパラメータブロックの先頭アドレス
パラメータブロック

オフセット	サイズ	内　容
00H	WORD	フォント種別
02H	BYTE	横書き／縦書きフラグ
03H	BYTE	動作指定フラグ
04H	WORD	X 方向ボディフェイスサイズ
06H	WORD	Y 方向ボディフェイスサイズ
08H	WORD	キャラクタフェイス開始 X 座標
0AH	WORD	キャラクタフェイス開始 Y 座標
0CH	WORD	X 方向キャラクタフェイスサイズ
0EH	WORD	Y 方向キャラクタフェイスサイズ
10H	16 バイト	未使用（常に 00H を設定）

●フォント種別

フォントの種別を指定します。

ただし、フォント種別に“3”か“4”が指定できるのは本体マルチフォントが実装されている場合のみです。また“3”はマルチフォント ROM ボードが PC-9801-38L でなければなりません。

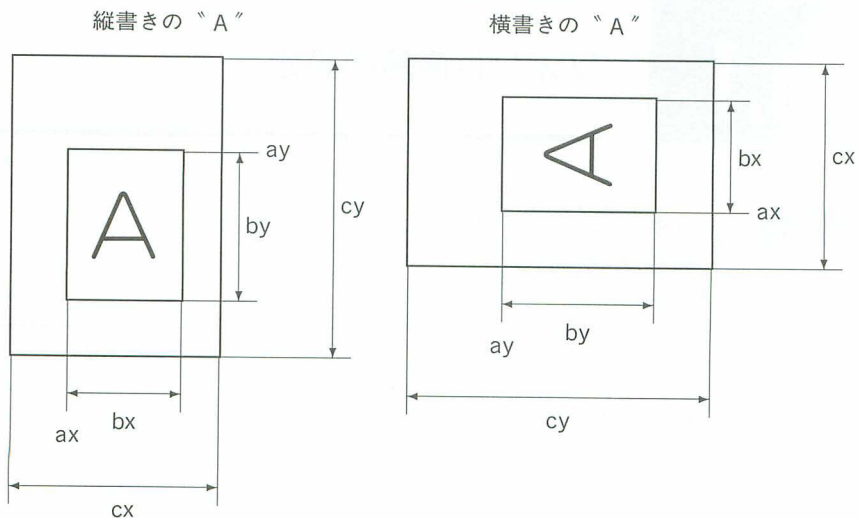
0　本体フォント ROM のフォントを利用
1　マルチフォント ROM ボードのゴシックフォントを利用
2　マルチフォント ROM ボードの明朝フォントを利用
3　マルチフォント ROM ボードの 16 × 16 ドットフォントを利用
4　本体マルチフォントの 20 × 20 ドットフォントを利用

●横書き／縦書きフラグ

フォントを縦書きにするか横書きにするかを指定します。

0　横書きフォント
1　縦書きフォント

縦書きフォントは、横書きフォント（テキストフォントと同じ向きのフォント）を反時計まわりに 90°回転した書体のフォントです。縦書きフォントの場合は、X 方向と Y 方向の各サイズが入れ換わることに注意してください。

例）　縦書きの〝A〞　横書きの〝A〞

ax：キャラクタフェイス開始 X 座標
ay：キャラクタフェイス開始 Y 座標
cx：X 方向ボディフェイスサイズ
cy：Y 方向ボディフェイスサイズ
bx：X 方向キャラクタフェイスサイズ
by：Y 方向キャラクタフェイスサイズ

●動作指定フラグ

フォント取得時の動作を指定します。

ただし、動作指定フラグに〝2〞を指定した場合、「フォント種別」の設定は無視されます。

0　8 × 8〜128 × 128 ドットフォントの取得
　（フォントドライバのバージョン 1.0 互換機能）
1　8 × 8〜400 × 400 ドットフォントの取得
2　利用者定義フォントを参照して拡大、縮小

● X 方向/Y 方向ボディフェイスサイズ

取得する文字フォントの X 方向/Y 方向ドット数を指定します。

有効値はフォントの種別によって次のようになります。

・フォント種別が 0 の場合

8 ≦ X 方向ボディフェイスサイズ≦ MIN（40、最大 X 方向ドット数） かつ
8 ≦ Y 方向ボディフェイスサイズ≦ MIN（40、最大 Y 方向ドット数）

ただし MIN（40、最大 X 方向ドット数）とは、〝40〞または〝最大 X 方向ドット数〞のいずれか小さい方を表します。

・フォント種別が1または2の場合
　16≦X方向ボディフェイスサイズ≦最大X方向ドット数 かつ
　16≦Y方向ボディフェイスサイズ≦最大Y方向ドット数

注意　X方向ボディフェイスサイズ＝0、またはY方向ボディフェイスサイズ＝0の場合はテキストの文字フォントと同一のサイズが指定されたものとみなします。
テキストの文字フォントサイズはノーマルモードとハイレゾリューションモードで異なるため〝フォントの取得〟の実行前には〝フォント情報の取得〟でフォントサイズを取得し、フォントデータ格納サイズを確認してください。この場合はテキストと同一のフォントサイズを利用するため、キャラクタフェイス開始X座標からY方向キャラクタフェイスサイズまでのパラメータは無視されます。

●キャラクタフェイス開始X座標/Y座標
ボディフェイス内のキャラクタフェイス開始X座標/Y座標を指定します。

● X方向/Y方向キャラクタフェイスサイズ
キャラクタフェイスのX方向/Y方向のドット数を指定します。

(キャラクタフェイス開始X座標/Y座標) ＋ (X方向/Y方向キャラクタフェイスサイズ) ≦ (X方向/Y方向ボディフェイスサイズ)

リターン　AX ＝0　正常終了
　　　　　　　≠0　異常終了

解　説

フォントの種別、フォントサイズなどの情報を指定します。

フォントの種別、各種フォントサイズなどはアプリケーションの起動直後では、先行のアプリケーションがこれらを変更している可能性があるので不定となります。フォントドライバを利用する場合は、まずファンクションNo.1（フォント情報の設定）を実行後、ファンクションNo.2（フォント情報の取得）を実行してください。

指定されたキャラクタフェイスサイズによって、編集の基本となるフォントが異なります。したがって、指定されたキャラクタフェイスサイズと基本フォントサイズの差が大きいほど文字の歪みが大きくなります。

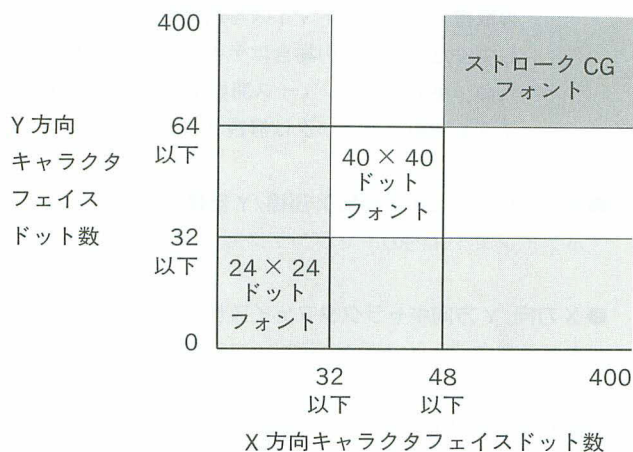
次に、指定されたキャラクタフェイスサイズと基本フォントの関係を説明します。

本体 CGROM のフォントを利用する場合

指定されたキャラクタフェイスサイズに関係なく、下記サイズのフォントを基本フォントとします。

ノーマルモード　　　　　　　：16 × 16 ドットゴシックフォント
ハイレゾリューションモード：24 × 24 ドット明朝フォント

マルチフォント ROM ボードのフォントを利用する場合

次の図のように指定されたキャラクタフェイスサイズにより、基本フォントが異なります。

マルチフォント ROM ボード（PC–9801–38L）の 16 × 16 ドットのフォントを利用する場合、指定されたキャラクタサイズに関係なく下記のサイズのフォントを基本フォントとします。

8 ≦ X 方向キャラクタフェイスサイズ≦ 40 の場合の基本フォント

ノーマル／ハイレゾリューションモード共に 16 × 16 ドットフォント

取得する文字コードが半角コード（2921H～2B7EH）であっても、X 方向ボディフェイスサイズ、Y 方向ボディフェイスサイズなどは全角コードの場合と同一で字体のみ半分になります。

本体マルチフォントからフォントを利用する場合

本体マルチフォントがある場合、マルチフォント ROM ボードがあっても 16 × 16、24 × 24、40 × 40 ドットのフォントは本体マルチフォントを優先して使用します。

キャラクタフェイスサイズによって取得される基本フォントはマルチフォント ROM ボードと同じサイズの基本フォントになります。

マルチフォント ROM ボードが実装されてストローク CG フォントを使用する場合、マルチフォント ROM ボードのストローク CG フォントを使用します。マルチフォント ROM ボードが実装されていない場合は、48 × 64 ドットまで拡大が可能です。

・16 × 16 ドットフォントを使用する場合

8 ≦ X 方向キャラクタフェイスサイズ≦ 40 の場合の基本フォント

ノーマル／ハイレゾリューションモード共に本体マルチフォントの 16 × 16 ドットフォント

・20 × 20 ドットフォントを使用する場合

8 ≦ X 方向キャラクタフェイスサイズ≦ 40 の場合の基本フォント

ノーマル／ハイレゾリューションモード共に本体マルチフォントの 20 × 20 ドットフォント

例）

ax：キャラクタフェイス開始 X 座標　　bx：X 方向キャラクタフェイスサイズ
ay：キャラクタフェイス開始 Y 座標　　by：Y 方向キャラクタフェイスサイズ
cx：X 方向ボディフェイスサイズ
cy：Y 方向ボディフェイスサイズ

INT CCH

ファンクション
No.2 フォント情報の取得

コール　スタック＝パラメータブロックの先頭アドレス

リターン　AX ＝0　常に正常終了
パラメータブロック

オフセット	サイズ	内　容
00H	WORD	フォント種別
02H	BYTE	横書き／縦書きフラグ
03H	BYTE	動作指定フラグ
04H	WORD	X 方向ボディフェイスサイズ
06H	WORD	Y 方向ボディフェイスサイズ
08H	WORD	キャラクタフェイス開始 X 座標
0AH	WORD	キャラクタフェイス開始 Y 座標
0CH	WORD	X 方向キャラクタフェイスサイズ
0EH	WORD	Y 方向キャラクタフェイスサイズ
10H	16バイト	未使用

解　説

現在のフォント種別、フォントサイズなどの情報を取得します。

パラメータブロックのフォント種別、横書き／縦書きフラグなどの説明についてはファンクション No.1（フォント情報）を参照してください。

INT CCH

ファンクション
No.3 フォントの取得

ファンクション No.1（フォント情報の設定）の「動作指定フラグ」の設定値により、各フォントの取得を行います。

1. 動作指定フラグに 0 が指定された場合

コール

スタック＝パラメータブロックの先頭アドレス
パラメータブロック

オフセット	サイズ	内　　容
00H	WORD	文字コード
02H	WORD	フォントデータ格納域サイズ
04H	DWORD	フォントデータ格納域アドレス
08H	24 バイト	未使用（常に 00H を設定）

●文字コード

取得する文字を 2 バイト日本語 JIS コードで指定します。2 バイト日本語 JIS コード体系は次の範囲で指定します。2 バイト日本語 JIS コード以外のコードが指定された場合はエラーになります。またマルチフォント ROM ボードからは、拡張文字（7921H〜7C7EH）の取得はできません。

2 バイト日本語 JIS コード体系……2121H〜7E7EH

●フォントデータ格納域サイズ

バイト単位でフォントデータ格納域サイズを指定します。指定する格納域サイズは次の条件を満たさなければなりません。

格納域サイズ≧（X 方向フォントサイズ＋7）¥8 × Y 方向フォントサイズ＋4

¥は商の整数部をとることを意味します。

ファンクション No.1（フォント情報の設定）で縦書きを指定した場合、X 方向/Y 方向のフォントサイズが入れ換わることに注意してください。

●フォントデータ格納域アドレス

フォントデータの格納域アドレスをダブルワードで指定します。

リターン　AX ＝0　正常終了
　　　≠0　異常終了

解　説

マルチフォント ROM ボード、または本体 ROM から現在のフォント情報をもとに文字フォントを取得します（バージョン 1.0 互換機能）。

また、フォントデータの格納形式は次のようになります。

オフセット	内容	
0	X 方向ボディフェイスサイズドット数（1 ワード）	Y 方向ボディフェイスサイズドット数（1 ワード）
$4+0\times\alpha$	Y 方向 1 ドット分の X 方向 m ドット分のパターン（1）	
$4+1\times\alpha$	Y 方向 1 ドット分の X 方向 m ドット分のパターン（2）	
$4+2\times\alpha$	⋮	
$4+n\times\alpha$	Y 方向 1 ドット分の X 方向 m ドット分のパターン（n）	
$4+(n+1)\times\alpha$		

m　：X 方向ボディフェイスサイズ
n　：Y 方向ボディフェイスサイズ
α　：(m+7) ¥8

メモリ内低位バイトから順に各ビットが X 座標のドット（昇順）に対応します。

例　本体 ROM フォントより 16×16 ドットフォントを取得する場合

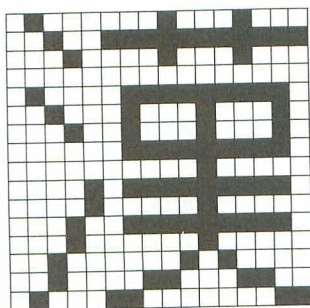
フォント情報の設定で指定するパラメータ

フォント種別	0
横書き／縦書きフラグ	0
動作指定フラグ	0
X 方向ボディフェイスサイズ	16
Y 方向ボディフェイスサイズ	16
キャラクタフェイス開始 X 座標	0
キャラクタフェイス開始 Y 座標	0
X 方向キャラクタフェイスサイズ	16
Y 方向キャラクタフェイスサイズ	16

フォントの取得で設定するパラメータ

文字コード	3441H
フォントデータ格納域サイズ	36
フォントデータ格納域アドレス	任意のアドレス

次に示すフォントイメージの場合、フォントデータは次のようになります。

フォントイメージ　　フォントデータ

← WORD →	オフセット	
0010H	0	X 方向ドット数
0010H	2	Y 方向ドット数
8840H	4	
FF27H	6	
8810H	8	
0000H	10	
:		
5020H	30	
8C21H	32	
0346H	34	

注意　動作フラグの指定で 0 を指定すると、バージョン 1.0 の互換動作となるため、CONFIG.SYS ファイルで/M スイッチをどのようにして指定しても、縦横それぞれ 128 ドットを超えるフォントの取得は行えません。

2. 動作指定フラグに 1 が指定された場合

コール　スタック＝パラメータブロックの先頭アドレス

パラメータブロック

オフセット	サイズ	内　容
00H	WORD	文字コード
02H	WORD	フォントデータ格納域サイズ
04H	DWORD	フォントデータ格納域アドレス
08H	WORD	作業域サイズ
0AH	DWORD	作業域アドレス
0EH	18 バイト	未使用（常に 00H を設定）

●文字コード、フォントデータ格納域サイズ、フォントデータ格納域アドレス

「1. 動作フラグに 0 が指定された場合」を参照してください。

●作業域サイズ

フォント取得のために必要な作業域を確保し、そのサイズを設定します。

必要な作業域のサイズは、取得するフォントのサイズから次のように算出します。

作業域サイズ≧（A＋7）¥8 × A＋15

¥は商の整数部をとることを意味します。

A には X 方向、Y 方向キャラクタフェイスサイズの値の大きい方を指定します。

例） 60 × 70 ドットフォントを取得するときのサイズ
作業域サイズ≧ 645 ＝（70＋7）¥8 × 70＋15

また、16 × 16、20 × 20、24 × 24、40 × 40 ドットフォントから基本フォントを取得するときの作業域サイズの算出方法は次のようになります。

作業域サイズ≧ W1＋W2
W1 ＝（B＋7）¥8 × B
W2 ＝ A¥8 × B

A は基本フォントサイズ（40、20、24、16）を指定します。

B には A（基本フォントサイズ）、X 方向、Y 方向キャラクタフェイスサイズの値の大きい方を指定します。

例） 30 × 45 ドットフォントを取得するときのサイズ
作業域サイズ≧ 405 ＝（45＋7）¥8 × 45＋24¥8 × 45

●作業域アドレス

利用者が確保した作業域の先頭をダブルワードで指定します。

解　　説

現在のフォント情報を基に文字フォントを取得します。

取得可能なフォントサイズは 8 × 8～400 × 400 ドットです。

3. 動作指定フラグに 2 が指定された場合

コール　スタック＝パラメータブロックの先頭アドレス

パラメータブロック

オフセット	サイズ	内　容
00H	DWORD	利用者定義フォントアドレス
04H	WORD	X 方向入力キャラクタフェイスサイズ
06H	WORD	Y 方向入力キャラクタフェイスサイズ
08H	WORD	フォントデータ格納域サイズ
0AH	DWORD	フォントデータ格納域アドレス
0EH	WORD	作業域サイズ
10H	DWORD	作業域アドレス
14H	12 バイト	未使用（常に 00H を設定してください）

●利用者定義フォントアドレス

ダブルワードで指定した利用者定義フォントアドレスには、次のようなフォーマットでフォントが格納されていなければなりません。

メモリ内低位バイト、MSB 側から順に各ビットが X 座標のドット（昇順）に対応

m：X 方向入力ドット数
n：Y 方向入力ドット数
α：(m ＋ 7)¥8

● X 方向/Y 方向入力キャラクタフェイスサイズ

入力するフォントサイズは 16 × 16〜64 × 64 の範囲で指定します。

●フォントデータ格納域サイズ

バイト単位でフォントデータ格納域サイズを指定します。指定する格納域サイズは次の条件を満たさなければなりません。

格納域サイズ≧（X 方向ボディフェイスサイズ＋7）¥8 × Y 方向ボディフェイスサイズ＋4

¥は商の整数部をとることを意味します。

●フォントデータ格納域アドレス

フォントデータの格納域アドレスの先頭をダブルワードで指定します。

指定した格納アドレスには、動作指定フラグに0または1を指定した場合と同じ形式で拡大／縮小されたフォントデータが格納されます。

●作業域サイズ

フォント取得のために必要な作業域を確保し、そのサイズを設定します。

必要な作業域のサイズは、取得するフォントのサイズから次のように算出します。

作業域サイズ≧ W1＋W2
W1＝（A＋7）¥8×A
W2＝（Y方向キャラクタフェイスサイズ＋7）¥8×A

¥は商の整数部をとることを意味します。

Aは X方向、Y方向入力キャラクタフェイスサイズ、X方向、Y方向キャラクタフェイスサイズの値の大きい方を指定します。

例）24×24ドットの利用者定義フォントから、30×40ドットフォントを取得するときの作業域サイズ

作業域サイズ≧ 320＝（40＋7）¥8×40＋（24＋7）¥8×40

●作業域アドレス

作業域の先頭アドレスをダブルワードで指定します。

解　説

利用者が定義したフォントイメージの拡大／縮小を行います。

6.5 エラーコード一覧

各ファンクションではリターン値が AX ≠ 0 のときは、エラーコードを返します。エラーコードとその意味については次のとおりです。

エラーコード	意　味
01	不正呼び出し。パラメータに誤りがあります。
02	CONFIG.SYS ファイルで定義した上限（最大ボディフェイスサイズ）を超えているため実行不可能です。
03	現在のハードウェアではこの機能は利用できません。
04	指定された文字コードは 2 バイト JIS コード体系です。
05	利用環境の設定により、ストローク CG フォントは利用できません。

6.6 プログラム例

次にマクロアセンブラ（MASM）によるプログラムの例を掲載します。

```
;
;       エントリテーブル内相対アドレスの定義
;
FONTVER         EQU    0             ; バージョンの取得
FONTSET         EQU    01*4          ; フォント情報の設定
FONTGET         EQU    02*4          ; フォント情報の取得
MOJIGET         EQU    03*4          ; フォントの取得
;
;       データ領域の定義
;
DATA            SEGMENT
FONT_ADDR       DD     0             ; エントリテーブル先頭アドレス格納域
FONT_PARA_ADDR  LABEL  DWORD         ; パラメータブロックアドレス格納域
FONT_PARA_OFF   DW     0             ; オフセットアドレス
FONT_PARA_SEG   DW     0             ; セグメントアドレス
FONT_PARA       DB     32  DUP(0)    ; パラメータブロック領域
FONT_PARA_SIZ   EQU    32            ; パラメータブロックサイズ
 :
 :
FONT_DATA       DB     2052 DUP(0)   ; フォントデータ格納域
DATA            ENDS
```

```
;
;        スタック領域の定義
;
STACK            SEGMENT
                 DW    256  DUP(0)
STACK            ENDS
;
CODE             SEGMENT
                 ASSUME  CS : CODE, DS : DATA
START :
                 MOV    AX, DATA
                 MOV    DS,AX
                 MOV    WORD PTR DS : FONT_PARA_SEG,AX
                 MOV    AX,OFFSET FONT_PARA
                 MOV    WORD PTR DS : FONT_PARA_OFF,AX
;        エントリテーブル先頭アドレスの取得
                 MOV    AX,0000H
                 MOV    BX,OFFSET DS : FONT_ADDR
                 INT    0CCH            ; エントリテーブル先頭アドレス
                                        ; の取得
;        バージョンの取得
                 LES    DI,DS : FONT_PARA_ADDR
                 PUSH   ES
                 PUSH   DI
                 LES    DI,DS : FONT_ADDR
                 CALL   DWORD PTR ES : FONTVER[DI]
;        フォント情報の設定
                 LES    DI,DS : FONT_PARA_ADDR
                 PUSH   DI
                 MOV    CX,FONT_PARA_SIZ/2
                 SUB    AX,AX
                 REP    STOSW           ; パラメータブロックのクリア
                 POP    DI
                 MOV    WORD PTR ES : 0[DI],0     ; フォント種別
                 MOV    BYTE PTR ES : 2[DI],0     ; 横書き／縦書きフラグ
                 MOV    WORD PTR ES : 4[DI],18    ;X 方向ボディフェイス
                                                  ; サイズ
                 MOV    WORD PTR ES : 6[DI],18    ;Y 方向ボディフェイス
                                                  ; サイズ
                 MOV    WORD PTR ES : 8[DI],2     ; キャラクタフェイス
                                                  ; 開始 X 座標
                 MOV    WORD PTR ES : 10[DI],2    ; キャラクタフェイス
```

```
                                                ；開始 Y 座標
          MOV   WORD PTR ES：12[DI],16  ;X 方向キャラクタ
                                        ；フェイスサイズ
          MOV   WORD PTR ES：14[DI],16  ;Y 方向キャラクタ
                                        ；フェイスサイズ
          PUSH  ES
          PUSH  DI
          LES   DI,DS：FONT_ADDR
          CALL  DWORD PTR ES：FONTSET[DI]
;     フォント情報の取得
          LES   DI,DS：FONT_PARA_ADDR
          PUSH  ES
          PUSH  DI
          LES   DI,DS：FONT_ADDR
          CALL  DWORD PTR ES：FONTGET[DI]
           ：
           ：
;     フォントの取得
          LES   DI,DS：FONT_PARA_ADDR
          PUSH  DI
          MOV   CX,FONT_PARA_SIZ/2
          SUB   AX,AX
          REP   STOSW                   ；パラメータブロック
                                        ；のクリア
          POP   DI
          MOV   WORD PTR ES：0[DI],3441H  ;2 バイト日本語
                                          ;JIS コード
          MOV   WORD PTR ES：2[DI],58     ；フォントデータ
                                          ；格納域サイズ
          MOV   AX,OFFSET DS：FONT_DATA
          MOV   WORD PTR ES：4[DI],AX
          MOV   WORD PTR ES：6[DI],DS     ；フォントデータ
                                          ；格納域アドレス
          PUSH  ES
          PUSH  DI
          LES   DI,DS：FONT_ADDR
          CALL  DWORD PTR ES：MOJIGET[DI]
           ：
           ：
CODE      ENDS
          END   START
```

索引

英数字

ア

カ

サ

タ

マ

ヤ

ラ

ワ